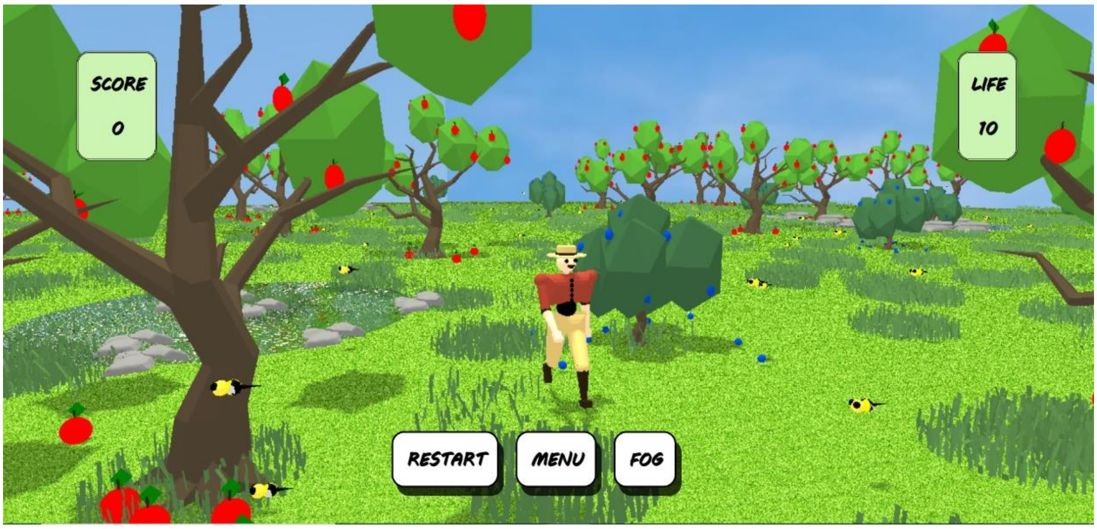
Interactive Graphic

Final Project Report – A.Y. 2021/2022

Francesca Andreotti

ID student number: 1696976

Pick Fruit!



- ## Introduction

  Pick Fruit! Is a single player game in which the objective is to reach the necessary score in order to win, without loose all the available life points.

  STORY

  The user has to bring the farmer in the orchard toward the blueberry's brambles and apples trees in order to pick as much as possible fruit. Unfortunately, life is not always so easy .... The user has to beware from bees which will try to sting the farmer if it get too close! Moreover, in the orchard, there are some lakes… and so there is the risk for the farmer of falling into the lakes!

MENU

The game start in menu page, in which you can find the resume of the game story and two clickable buttons:

- START GAME, for starting directly to play
- HOW TO PLAY, which brings you to another page in which are explained the game rules

HOW TO PLAY

When you start the game, you can move the farmer in order to reach the fruit and exploring the map using the keyboard arrows

- **Arrow up** moves the farmer forward;
- **Arrow left/right** change the direction of farmer motion, rotating him respectively to left and right

During the game, the user can change the camera view by using the mouse.

Moreover, for increasing the difficulty of the game, it is possible to add fog to the scene, by clicking the button FOG

If you want to restart the game, it is sufficient to press RESTART button

GENERAL RULES

In the upper part of the screen, there are **two point counters**:

- LIFE (right upper corner), which represent farmer life points. It starts from 10 points and decreases of 1 each time a bee stings you

- SCORE (left upper corner), which represent the fruit quantity taken under the threes. It starts from 0 points and increases of 1 each time you approach a fruit on the ground

When the farmer comes close the fruit which is on the ground close to threes, the score counter increases, showing a message which tells that some fruits it is taken. Instead, when the character hit a bee, it stings him, causing the decreasing of the life counter. If the farmer goes beyond the shore of one of the lakes, all the life points over.

HOW QUIT THE GAME

There are two possible endings:

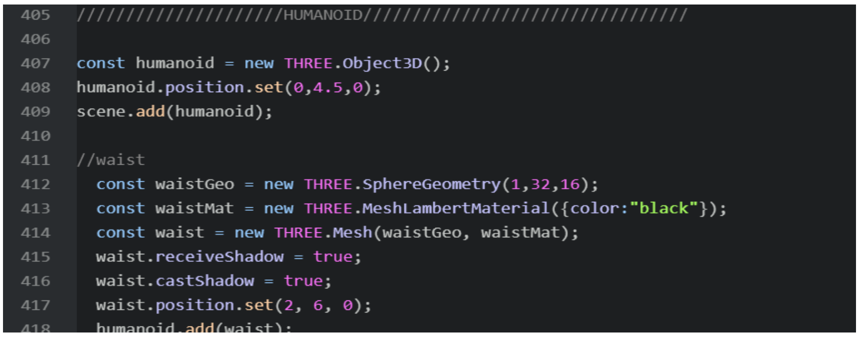
- if SCORE counter reaches 20 points, then you win the game
- if LIFE counter reaches 0 points OR you fall in a lake, then you loose the game

- ## Hierarchical models.

Almost all of the 3D models in the project are made in JS code, directly using THREE.JS library:

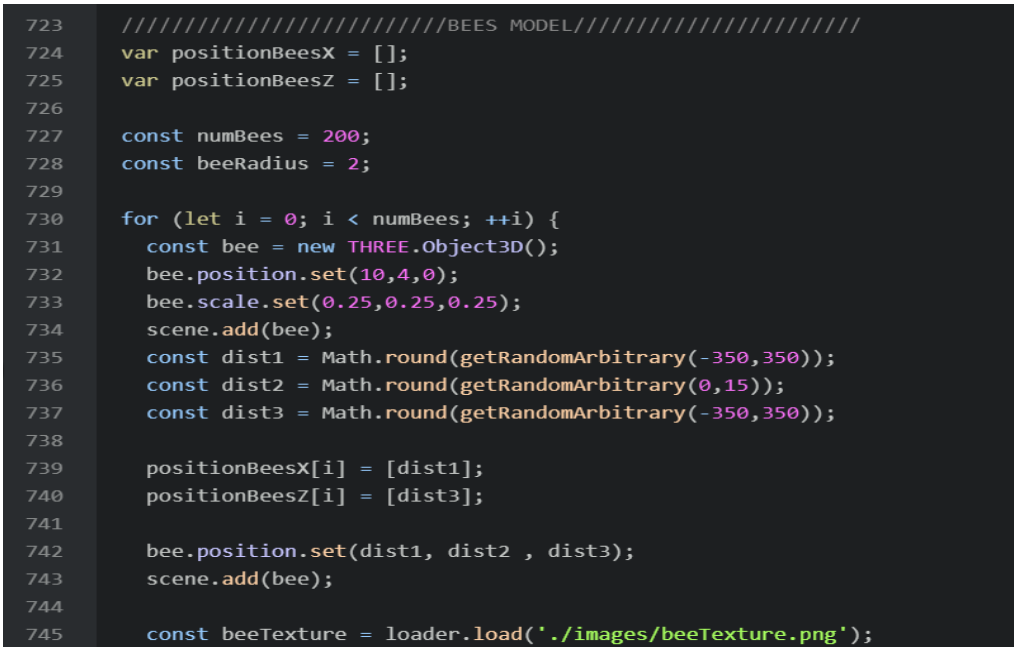
<u>Animated models</u>

- "humanoid" Object3D is a complex model composed by different Meshes which represents a **humanoid structure** which is composed by all the main human body parts and joints. The realization of this model and its animations were been the main objective of the project and the more challenging part.



```
////////////////////HUMANOID////////////////////////////////

const humanoid = new THREE.Object3D();
humanoid.position.set(0,4.5,0);
scene.add(humanoid);

//waist
  const waistGeo = new THREE.SphereGeometry(1,32,16);
  const waistMat = new THREE.MeshLambertMaterial({color:"black"});
  const waist = new THREE.Mesh(waistGeo, waistMat);
  waist.receiveShadow = true;
  waist.castShadow = true;
  waist.position.set(2, 6, 0);
  humanoid.add(waist);
```

- ➢ “bee” Object3D is a model representing a bee and its wings. In the implementation, many of them are rendered in order to build the game structure.

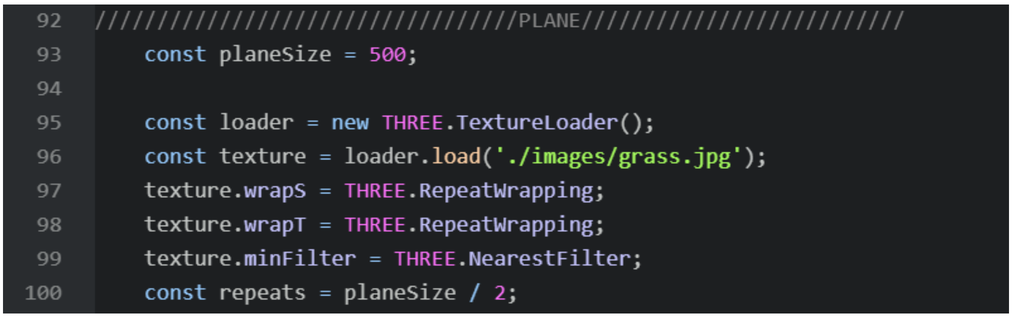


```
//////////////////////////BEES MODEL///////////////////////
var positionBeesX = [];
var positionBeesZ = [];

const numBees = 200;
const beeRadius = 2;

for (let i = 0; i < numBees; ++i) {
  const bee = new THREE.Object3D();
  bee.position.set(10,4,0);
  bee.scale.set(0.25,0.25,0.25);
  scene.add(bee);
  const dist1 = Math.round(getRandomArbitrary(-350,350));
  const dist2 = Math.round(getRandomArbitrary(0,15));
  const dist3 = Math.round(getRandomArbitrary(-350,350));

  positionBeesX[i] = [dist1];
  positionBeesZ[i] = [dist3];

  bee.position.set(dist1, dist2 , dist3);
  scene.add(bee);

  const beeTexture = loader.load('./images/beeTexture.png');
```

Environment models

- ➢ “sun” model, representing the sun in the scene

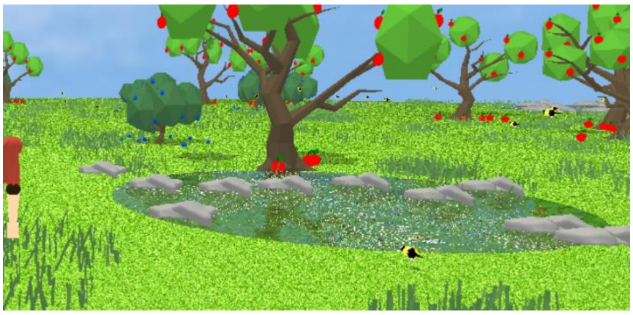
- ➢ “plane” model, representing the grass field (by setting a texture) on which the game takes place

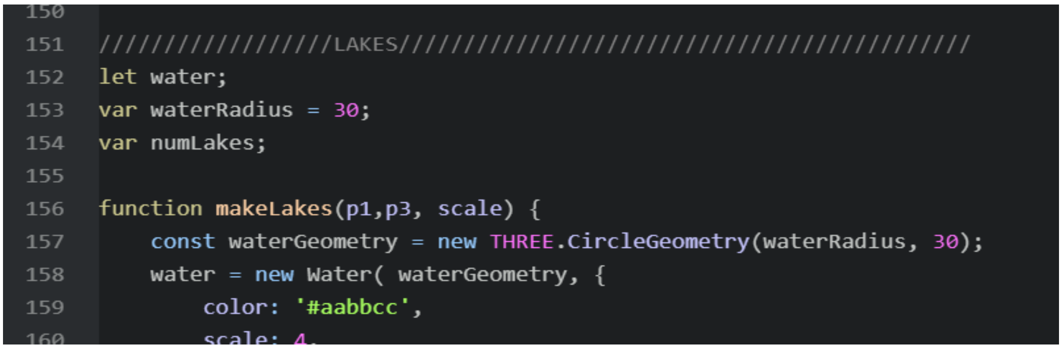


```
///////////////////////////////////PLANE//////////////////////////
    const planeSize = 500;

    const loader = new THREE.TextureLoader();
    const texture = loader.load('./images/grass.jpg');
    texture.wrapS = THREE.RepeatWrapping;
    texture.wrapT = THREE.RepeatWrapping;
    texture.minFilter = THREE.NearestFilter;
    const repeats = planeSize / 2;
```

- “sky” model, which is a sphere geometry representing the sky in the scene and containing the whole seen itself

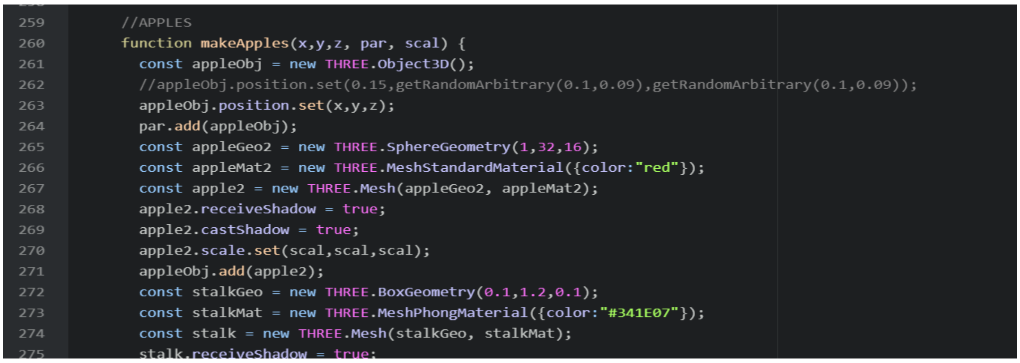
- “lakes” model, which is made by using makeLakes() function. In order to mimic the water behaviour, in the function is included the water model included in THREE.JS library. The bottom of the lake is made by using a texture and some rocks (obj model) are added randomly.





```
///////////////////LAKES/////////////////////////////////////////////////
let water;
var waterRadius = 30;
var numLakes;

function makeLakes(p1,p3, scale) {
    const waterGeometry = new THREE.CircleGeometry(waterRadius, 30);
    water = new Water( waterGeometry, {
        color: '#aabbcc',
        scale: 4,
```

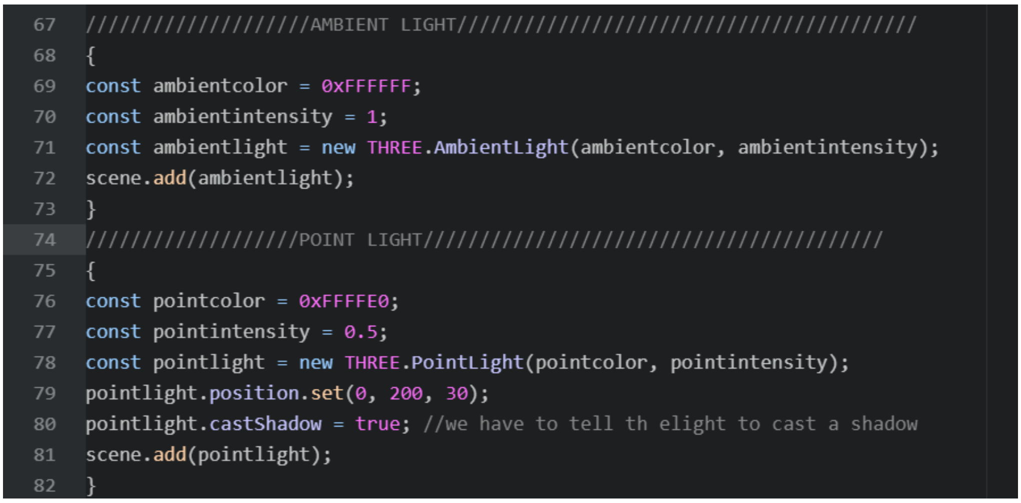
- In the scene, are implemented two three models: apples tree and blueberry brambles. Both models use the tree.obj in order to implement the trunk, while the leaves and the fruits on the branches and on the ground are models which are made directly as 3D objects. In particular, the apples are composed by a leaf and the stalk meshes. Apples and blueberryes models are implemented respectively in makeApples() and makeBluberries() functions.



```
    //APPLES
    function makeApples(x,y,z, par, scal) {
      const appleObj = new THREE.Object3D();
      //appleObj.position.set(0.15,getRandomArbitrary(0.1,0.09),getRandomArbitrary(0.1,0.09));
      appleObj.position.set(x,y,z);
      par.add(appleObj);
      const appleGeo2 = new THREE.SphereGeometry(1,32,16);
      const appleMat2 = new THREE.MeshStandardMaterial({color:"red"});
      const apple2 = new THREE.Mesh(appleGeo2, appleMat2);
      apple2.receiveShadow = true;
      apple2.castShadow = true;
      apple2.scale.set(scal,scal,scal);
      appleObj.add(apple2);
      const stalkGeo = new THREE.BoxGeometry(0.1,1.2,0.1);
      const stalkMat = new THREE.MeshPhongMaterial({color:"#341E07"});
      const stalk = new THREE.Mesh(stalkGeo, stalkMat);
      stalk.receiveShadow = true;
```

- **Lights and shadows**

  - The lights implemented are two: an ambient light and a point light, miming the behaviour of the sun light

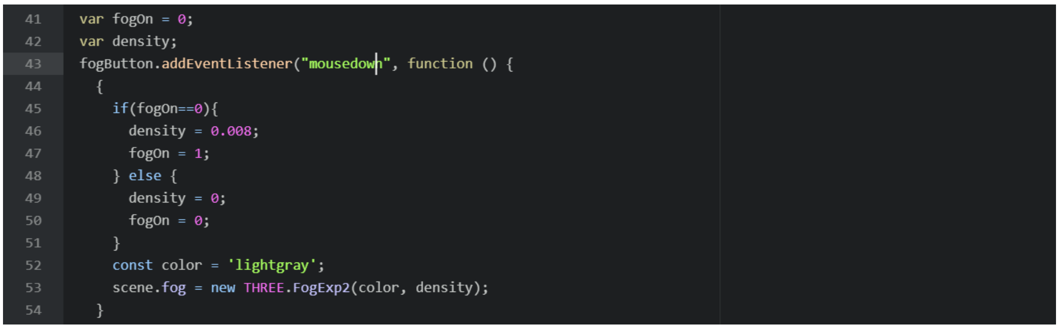


```
///////////////////AMBIENT LIGHT///////////////////////////////////////
{
const ambientcolor = 0xFFFFFF;
const ambientintensity = 1;
const ambientlight = new THREE.AmbientLight(ambientcolor, ambientintensity);
scene.add(ambientlight);
}
/////////////////POINT LIGHT//////////////////////////////////////
{
const pointcolor = 0xFFFFE0;
const pointintensity = 0.5;
const pointlight = new THREE.PointLight(pointcolor, pointintensity);
pointlight.position.set(0, 200, 30);
pointlight.castShadow = true; //we have to tell th elight to cast a shadow
scene.add(pointlight);
}
```

  - In particular, the point light is implemented in order to cast shadows in all the scene. All the models which are implemented in the project are set in order to receive and cast shadows

- **Fog Option**

During the game, the user can choose if add in the scene some fog, in order to increase the difficult of the game



```
var fogOn = 0;
var density;
fogButton.addEventListener("mousedown", function () {
  {
    if(fogOn==0){
      density = 0.008;
      fogOn = 1;
    } else {
      density = 0;
      fogOn = 0;
    }
    const color = 'lightgray';
    scene.fog = new THREE.FogExp2(color, density);
  }
```

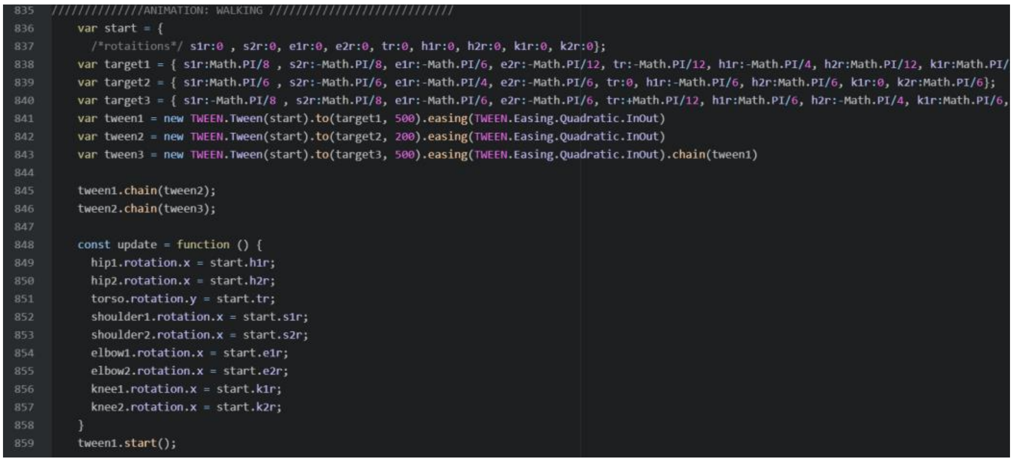
- ## Camera

The camera implemented is a perspective one. By using OrbitsControl.js library, during the play it is let the user to spin or to orbit the camera around the farmer character. This function is implemented by updating the camera controls and the target position in the render() function, in order to follow properly the farmer movement during the changes of its position.

- ## Animations

Most of the objects are animated and all the animation are implemented by using TWEEN.JS library.

The humanoid performs walking animation, moving the rigid body components by rotating the various joints which represent the human articulations (ankles, wrists, elbows and hips). For smoother and realistic animation, are used different tweens (chain of tweens) and Quadratic easing function (InOut mode).

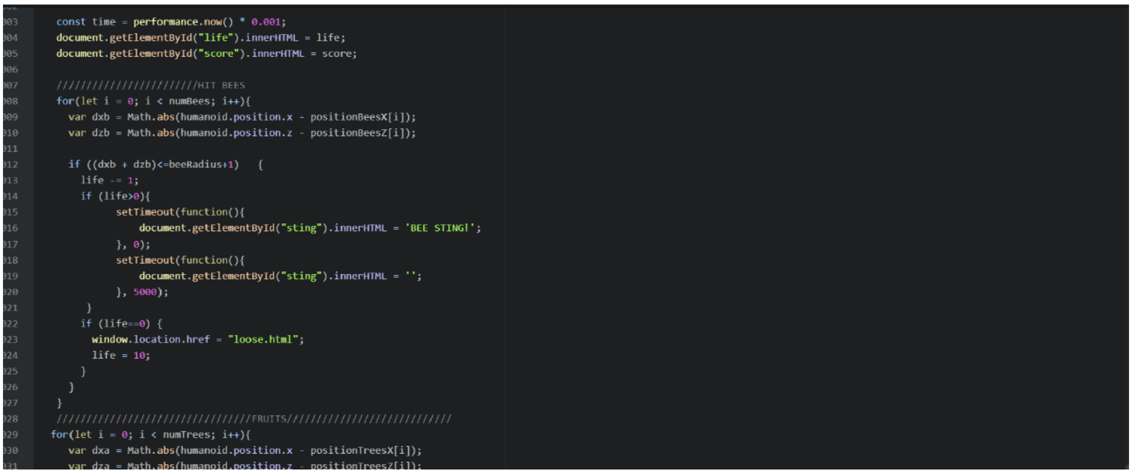


```
//////////////ANIMATION: WALKING ////////////////////////////
    var start = {
      /*rotaitions*/ s1r:0 , s2r:0, e1r:0, e2r:0, tr:0, h1r:0, h2r:0, k1r:0, k2r:0};
    var target1 = { s1r:Math.PI/8 , s2r:-Math.PI/8, e1r:-Math.PI/6, e2r:-Math.PI/12, tr:-Math.PI/12, h1r:-Math.PI/4, h2r:Math.PI/12, k1r:Math.PI/
    var target2 = { s1r:Math.PI/6 , s2r:-Math.PI/6, e1r:-Math.PI/4, e2r:-Math.PI/6, tr:0, h1r:-Math.PI/6, h2r:Math.PI/6, k1r:0, k2r:Math.PI/6};
    var target3 = { s1r:-Math.PI/8 , s2r:Math.PI/8, e1r:-Math.PI/6, e2r:-Math.PI/6, tr:+Math.PI/12, h1r:Math.PI/6, h2r:-Math.PI/4, k1r:Math.PI/6,
    var tween1 = new TWEEN.Tween(start).to(target1, 500).easing(TWEEN.Easing.Quadratic.InOut)
    var tween2 = new TWEEN.Tween(start).to(target2, 200).easing(TWEEN.Easing.Quadratic.InOut)
    var tween3 = new TWEEN.Tween(start).to(target3, 500).easing(TWEEN.Easing.Quadratic.InOut).chain(tween1)

    tween1.chain(tween2);
    tween2.chain(tween3);

    const update = function () {
      hip1.rotation.x = start.h1r;
      hip2.rotation.x = start.h2r;
      torso.rotation.y = start.tr;
      shoulder1.rotation.x = start.s1r;
      shoulder2.rotation.x = start.s2r;
      elbow1.rotation.x = start.e1r;
      elbow2.rotation.x = start.e2r;
      knee1.rotation.x = start.k1r;
      knee2.rotation.x = start.k2r;
    }
    tween1.start();
```

The bee model is animated in the same way, but this time they go up and down, moving their wings.

- ## Score/life System implementation

The assignment and remotion of point (score and life) is implemented during render time, in render() function, by increasing/decreasing counters and sending results and consequent messages to the HTML file



```
    const time = performance.now() * 0.001;
    document.getElementById("life").innerHTML = life;
    document.getElementById("score").innerHTML = score;

    ////////////////////////HIT BEES
    for(let i = 0; i < numBees; i++){
      var dxb = Math.abs(humanoid.position.x - positionBeesX[i]);
      var dzb = Math.abs(humanoid.position.z - positionBeesZ[i]);

      if ((dxb + dzb)<=beeRadius+1)   {
        life -= 1;
        if (life>0){
              setTimeout(function(){
                  document.getElementById("sting").innerHTML = 'BEE STING!';
              }, 0);
              setTimeout(function(){
                  document.getElementById("sting").innerHTML = '';
              }, 5000);
         }
        if (life==0) {
          window.location.href = "loose.html";
          life = 10;
        }
      }
    }
    ////////////////////////////////FRUITS//////////////////////////
   for(let i = 0; i < numTrees; i++){
      var dxa = Math.abs(humanoid.position.x - positionTreesX[i]);
      var dza = Math.abs(humanoid.position.z - positionTreesZ[i]);
```

External Objects

The few external objects which are not implemented in the JS file are the following:

- Stones.obj (which are the rocks in the water lakes)
- Tree.obj (which is the trunk of the trees)
- Grass.obj (which are the blades of grass randomly putted in the scene)

The objects are imported and loaded using OBJloader.js, which is included in three.js library. They are taken from Clara.io website.

Libraries and References

- three.js
- tween.js
- https://threejs.org/manual/
- https://threejs.org/

Personal Conclusions

Since this was the first time for me about implementing such kind of application, it has been very challenging at the beginning to take experience in programming this game. Instead now I am very glad to have taken part to this course, because I have increase a lot my programming knowledge. I will go on working on this application; in fact it has to be still improved.