CHILL & PLAY

Interactive Graphics

Final Project

A.Y. 2021/2022

SAPIENZA
UNIVERSITÀ DI ROMA

Mario Cosimo Angelini 2029254
Laura Papi 1760732
Francesca Restante 1745106
Daniela Rieti 1762973

Index

1. Introduction

1.1 The idea

For this project we wanted to create an eye catching digital experience, therefore we offer the user simple designs and a minimalistic but attractive environment.

In this way the user can take his mind off for a while and relax with our mini games.
In fact, both the games are designed like sandboxes in which the user doesn't have to win or lose, but he just interacts in a peaceful but entertaining game flow.

1.2 How to play

First of all the user has to choose the environment he wants to play in, selecting the frog or the sheep, and clicking on the go button. Now he can start the game.

Choose the **frog** if you want to play as a fly that bothers it while trying to not get eaten.
Move the mouse to pull the fly around the screen. You can pass through different game areas, in which the fly is always followed by the frog's gaze. The closer you get to the frog, the more dangerous it becomes.

- *Yellow area*: it is the farest from the frog and this means that when the fly is here, the frog can't reach it.
- *Orange area*: here the frog tries to reach the fly but he isn't yet able to catch it.
- ***Red area***: here the frog can catch the fly and eat it.

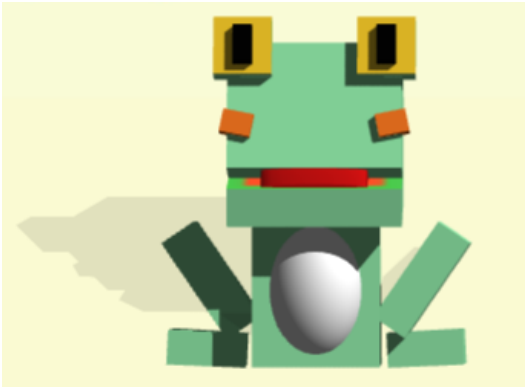
The game restarts automatically after the frog catches the fly and after a few seconds a new one appears.

Choose the **sheep** if you want to feel like a farmer who needs to shear his sheep. Like a real farmer, grab the scissors and move them over the sheep to remove the wool tufts. Once the wool is removed it falls on the ground and disappears. You can click and drag, or use the arrow keys, to rotate the sheep to not miss any spot.
When the sheep is completely sheared, she will be happy and grateful. Now you can choose to restart the game, or move back to the homepage.

2. Models

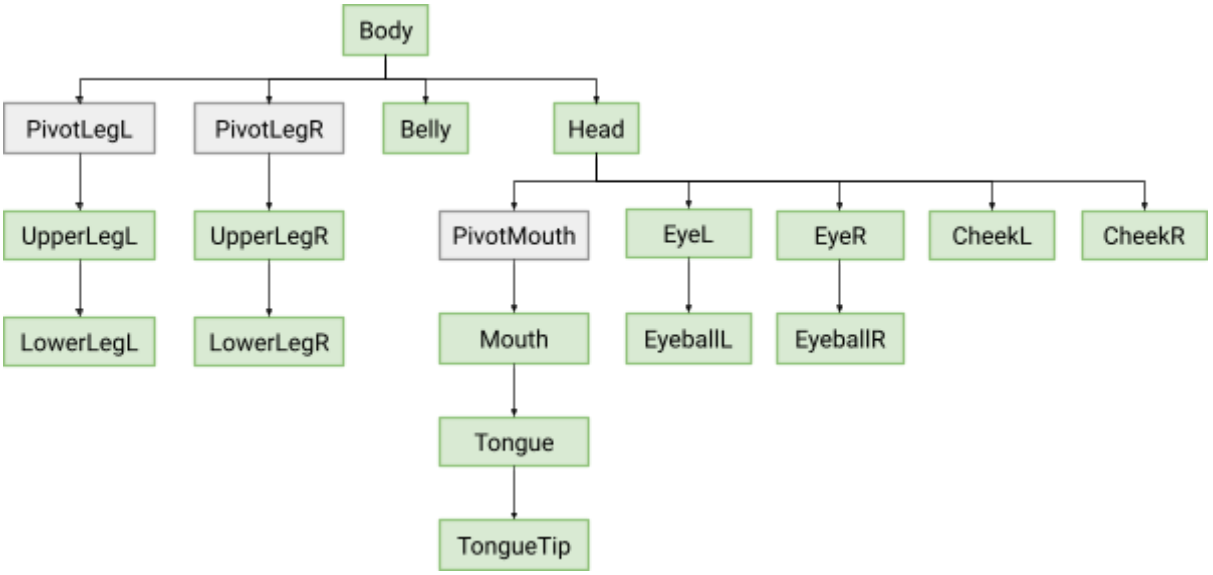
In the following chapter we will see how we structured each hierarchical model.
We chose to design all of our models by using a very cartoony style, using only simple and geometrical shapes combined together to create all the objects we needed.

2.1 Frog



To design the model of the frog we first started by drawing it, and then thinking about what kind of movements we needed it to do.
As we will see in the dedicated animation chapter, the frog will have to move its legs and mouth, and this is the reason why we chose to add pivot points in the hierarchical model.

The hierarchical tree appears as follows:



2.1.1 Body

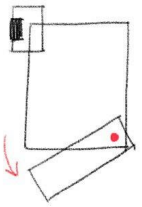
The body is the root node of the frog's hierarchical model, and so all the other components depend on it. To model the body we started from a BoxGeometry from three.js, and then we proceeded by modifying the position of each vertex, in order to obtain a truncated pyramid with a square base.
On the front face of the body we also decided to add a stretched SphereGeometry, representing the belly of the frog.

2.1.2 Head

The head of the frog, like all the connected components (eyes, eyeballs and cheeks) are created using BoxGeometry from three.js. The head is attached to the body, and translated to appear just above it. Eyes, eyeballs and cheeks differ in dimension parameters, and in the color of the material.

2.1.3 Mouth

To create the mouth we decided to simply add another BoxGeometry right below the head, representing the jaw.



Since we actually needed the frog to open the mouth, we created an Object3D element acting as a pivot point, in the exact position where the joint between jaw and head would be, and then we added the jaw as a child of the pivot element.

In this way rotating the pivot, would end up in rotating also the jaw, giving the effect of the mouth opening around the joint, like we see in the picture.

2.1.4 Tongue

The tongue is the frog's most complex component. In order to give it a curved shape we used the TubeGeometry from three.js, which creates an empty tube following the path defined by a CustomSinCurve.

First of all the path is defined by writing functions for all the axes:

```
const tx = t * 1 - 1.5;
const ty = 0.2 * Math.sin(3 * Math.PI * t );
const tz = 0;
```

and then the geometry is created and associated with that path:

```
const tongueGeometry = new THREE.TubeGeometry(tonguePath, 100, 1.1, 20, false);
```

The mesh is then scaled along the z axis in order to make it look flat, and not perfectly circular like the TubeGeometry.

Since the TubeGeometry creates an empty tube, it has open extremities, and to solve this problem we added another component (tongueTip) as a child of the tongue.

It is a very stretched sphere that perfectly covers the opening, and creates a flat and round tip for the tongue.

2.1.5 Legs

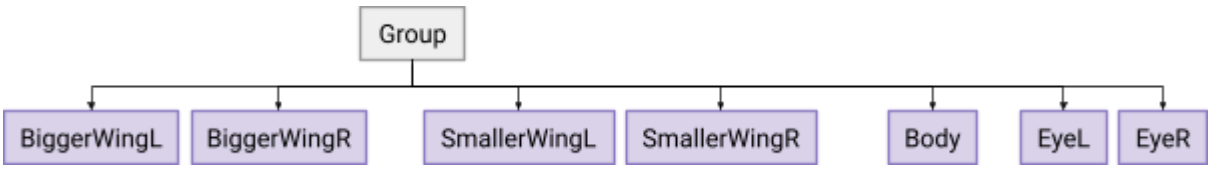
The legs are composed of two elements each, an upper component for the leg and a lower one for the foot. All of these parts are realized using BoxGeometry.

The important thing to notice is that they are not directly attached to the body, but the parent node for each leg is a pivot, connecting the leg to the body.

This is necessary to implement the animation, which is described in detail in the dedicated chapter.

2.2 Fly

The design idea behind the fly was to shape the body as a stretched sphere, the eyes as perfect spheres, and then create two sets of wings with elliptical shape.

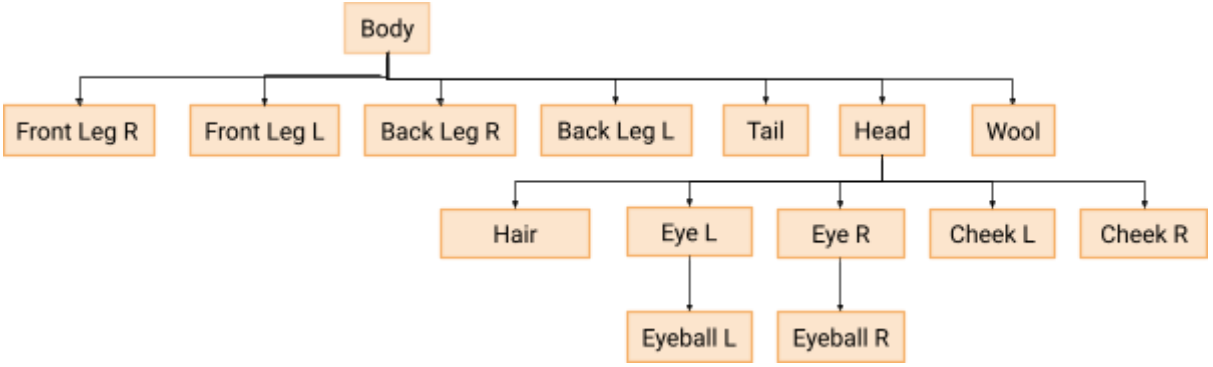


In order to create the stretched sphere for the body, we had to start from a SphereGeometry, and then scale the z axis to extend it. But using this method causes all the children of the body to be deformed as well.
To avoid this we managed to create a common father for all the elements of the fly (exploiting the Group constructor of THREE.js), such that we would be able to both move the fly around as a single hierarchical model, but also to scale and deform each single element independently from the others.

The same scaling method was also used to shape each wing, starting from a very short CylinderGeometry, and stretching its base.

2.3 Sheep

To design the model of the sheep we take as reference the hierarchical structure illustrated below. We started with the body and added the components gradually.
As we will see later, the movements of the model consist of rotating the eyes and head to the right and left and shaking its wool.



2.3.1 Sheep body and wool

The model of the sheep is mainly composed of a solid with twenty faces, using the class IcosahedronGeometry(radius : Float, detail : Integer) from the library THREE.js.

```
const sheepBodyGeometry = new THREE.IcosahedronGeometry(0.5, 0);
```

Like the body and the tail of the sheep, also the wool is made with the same geometry. Thanks to several for-loops it was possible to place different circles of wool around the body of the sheep. All the elements composing the wool are stored in an array wool[].
The head and tail use a different type of material and color.

2.3.2 Head

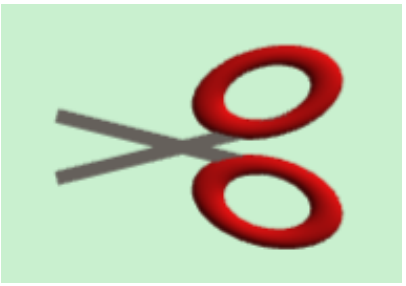
One of the fundamental elements of the sheep is its head. In this case the shape is given by a scaled and rotated icosahedron. It is attached to the body and positioned in the upper front part.
Eyes and eyeballs are created using a cylinder with eight segments, exploiting the CylinderGeometry class, while the cheeks are created with the SphereGeometry class.
The eyes are disposed symmetrically regarding the face of the sheep and just above there are the eyeballs. Below them there are the cheeks which are shifted slightly towards the border of the face. Each one of them has different colors and dimensions.

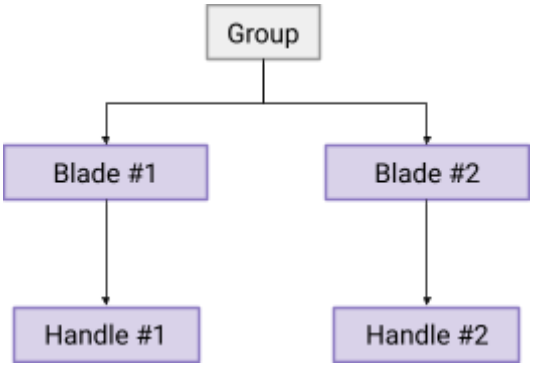
2.3.3. Legs

In creating the sheep's legs, a cylinder with 4 faces was used to give the effect of a parallelepiped. The material used is the same as the head. The legs were placed symmetrically on the four sides of the lower part of the body.

2.4 Scissors

The model used for the scissors is quite simple, it consists of a pair of blades and a pair of handles connected to each other. The group is used for the same reason as the fly. The handles are made through the TorusGeometry class, they are inclined in opposite directions with respect to the y axis and connected directly to the blades, the latter are composed of an elongated boxGeometry block and made of a gray material.





3. Lights and Materials

3.1 Lights and shadows

The project uses two types of lights: an ambient light and a directional light.
Both are initialized and created as soon as the homepage is loaded and also used in the rest of the scenes. The function that allows us to create them is:

```
function createLights(){
  const ambientLight = new THREE.AmbientLight(0xffffff, 0.3);
  scene.add(ambientLight);
  const light = new THREE.DirectionalLight( 0xffffff, 0.8);
  light.position.set( 200, 200, 200);
  light.castShadow = true;
  light.shadowDarkness = 0.2;
  light.shadow.radius = 8;
  light.shadow.mapSize.width = 2048; // default
  light.shadow.mapSize.height = 2048; // default
  scene.add( light ); }
```

Both lights are saved in some variables and are then added to the scene.
The first is used to illuminate the whole environment and give depth to the various models, while the second is used to make it clear on what level the animals are positioned, thus creating a shadow behind them.

The shadow is adjusted through the attribute of light castShadow, through which it is also possible to set the darkness and the range of action.

3.2 Materials

All the elements that are part of the scene are created using Mesh or MeshStandardMaterial to ensure the right lighting and a better view of the objects through the shadows. To enable shadows on each element we need to enable the castShadow attribute on that as well.

A particular case of material is given by the texture of the sheep, which uses a normal map.
We take as input from the textureLoader and exploit the property normalMap of the MeshStandardMaterial. The variable furNormalMap is initialized before and affects the surface normal for each pixel fragment, changing the way the color is lit.

```
const sheepBodyMaterial = new THREE.MeshStandardMaterial({
  color: 0xf3f2f7,
  normalMap: furNormalMap });
```

4. Animations

All the animations are created using the set of libraries made available by createJS. Here we will see a description of all the animations developed for each hierarchical model.

4.1 Frog

The frog model appears both in the homepage and in the dedicated frog page. The animations for each page are different, and they are described separately in the following sections.

4.1.1 Homepage

In the homepage we designed small, discreet, natural animations, just to give the idea that the frog is waiting for the game to start.

Eyes

First of all we wanted it to blink, and in order to do so we implemented an animation that scales the size of the pupils making them look like they are closing and opening.

```
initialY = frogPupilL.scale.y;
frogPupilLAnimation = createjs.Tween.get(frogPupilL.scale, {loop:true})
    .to({ y: 0.1 }, 100, createjs.Ease.linear)
    .to({ y: initialY }, 100, createjs.Ease.linear)
    .wait(2500);
```

These lines of code make the size of the pupil reduce to 0.1 in 100ms and then go back to their initial size. This happens in a loop every 2500ms thanks to the wait function at the end.

Mouth

The frog also slightly opens and closes the mouth, and this is implemented similarly to the previous case, but instead of changing a scale, now we rotate the mouth pivot introduced in the Models chapter.

Belly

We also decided to animate the belly of the frog, creating a movement as if the belly was grumbling. To give this illusion, a very fast and very small movement was implemented, slightly scaling the size of the belly up and down really fast.

Legs

To animate the legs we exploited the second pivot introduced in the previous chapter, that allows the upper legs to rotate around the joint.

But rotating the upper legs results in also rotating the feets (that are children of the upper legs), so to keep the feet firm on the ground we had to create an opposite synchronized animation for them.

4.1.2 In game

Jump

In order to make the frog jump, we combined an animation for the legs, very similar to the one described in the home section, and an animation for the body, that translates it up and down.

Notice that in this case we used createjs.Ease.circOut instead of linear to obtain a more natural movement. The circOut ease equation makes the movement slow down as the frog moves higher, imitating the effect of gravity.

Eyes

We wanted the frog to follow the fly around the screen with its eyes, and so move its pupils inside the eyes.

The idea behind this animation is to create a mapping between the scene's visible area, and the frog's eye area. In this way we are able to map the position of the fly on the screen, and translate it into a position of the pupil in the eye.

To achieve this we used the following formula:

```
var eyePos = {x: (((flyPos.x)*0.2)/width), y:(((flyPos.y)*0.2)/height)+2};
```

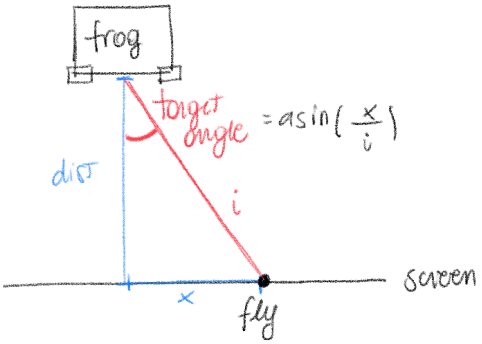
where flyPos is the position of the fly in world coordinates, while width and height are the dimensions of the scene's visible area, dynamically computed, so that they adapt to the dimensions of the window when it is resized by the user.

Head

The head needs to follow the fly too, rotating around the y axis (left and right) and around the x axis (up and down).

To compute the correct angle of rotation we need to look at the fly position, and transform it into the right rotation angle for the head.

An intuition is given by the schema at the right, representing the scene seen from the top.



This translates into the following formulas:

```
var dist = flyPos.z - frogHead.position.z;
var i = Math.sqrt(Math.pow(dist, 2)+Math.pow(flyPos.x, 2));
```

```
var targetAngle = Math.asin(flyPos.x/i);
frogHeadHorizontalAnimation = createjs.Tween.get(frogHead.rotation)
    .to({y: targetAngle }, 80, createjs.Ease.linear);
```

An analogue computation is done for the vertical movement (around x axis).

Catching the fly

When the fly stops for more than a second, the frog starts a sequence of animations to catch it:

1. The head tilts back by an angle computed to be proportional to the targetAngle described above. At the same time the mouth opens a bit, using the mouthPivot as discussed before.
2. The tongue is scaled and translated, to obtain the effect of a stretching tongue, in the direction of the fly. To better point at the fly, the tongue is also slightly rotated by an angle proportional to the targetAngle.

First case: eating the fly

If the fly stopped in the danger area, the frog will catch it and eat it, as described by the following steps:

3a. Once the tongue has reached the fly, the animations at step 1. and 2. are reverted: the tongue regains its original dimensions, and the head will look on the front, facing the camera.
Also, the fly dimensions are scaled down until it becomes so small that it's not visible anymore. And at the same time it is translated towards the mouth of the frog.

4a. Now that the fly was eaten by the frog, the last animation starts, where the frog, still facing the camera, blinks and then shakes its belly, to indicate that it has finished eating.

Second case: can't catch the fly

Instead, if the fly stopped in the warning area, the frog will try to reach it, but without succeeding:

3b. The tongue regains its original dimensions, the head is positioned facing the camera, same as in step 3a. But since the fly is not affected, it continues to move along with the mouse.

4b. Once the frog has retracted the tongue, without catching the fly, it proceeds by front facing the camera, and shaking the head (rotating it right and left).

An important thing to notice is that in order to execute these animations, we first had to cancel the requestAnimationFrame that was in charge of moving the head and the eyes following the mouse movements. The requestAnimationFrame is then restored at the end of the eating process, thanks to a timer.

4.2 Sheep

The sheep model is animated on the homepage and on its dedicated page. The animations for each page are different, and they are described separately in the following sections.

4.2.1 Homepage

Head and Eyes

The main movement the sheep makes on the homepage is to look around. To do so we took advantage of the rotation of the head on the x axis, followed by a translation of the pupils in the eye area. It is important to note that the head remains fixed for a few seconds to give a more realistic effect to the gaze.

Wool

After looking around, you can see a quick rotation of the tufts of wool as if the sheep were shaking. Thanks to the tween performed along the entire wool array, each element is rotated in both directions along the z axis.

4.2.2 In game

Once we enter the sheep page, we can see that the sheep continues to perform the same movements as the homepage in addition to other interactions.

Wool

When the mouse (to which the scissors are attached) passes over a tuft of wool, the latter will fall to the ground with a bounce. This is possible thanks to the bounceOut function, passed in the animation lines, as we can see below:

```
woolFalling = createjs.Tween.get(wool[j].position, {loop: false})
        .to({ y: -0.7}, 2000, createjs.Ease.bounceOut)
```

Once dropped to the ground, the wool disappears to allow the player to cut all the other tufts and better visualize the whole sheep:

```
woolSmall = createjs.Tween.get(wool[j].scale, {loop:false}).wait(2000)
        .to({x: 0.001, y: 0.001, z: 0.001}, 2000, createjs.Ease.bounceOut)
```

During the animation a variable takes into account how many tufts have been shaved and how many still remain.

Final Jump

Finally, once all the wool has been removed, the functions **resetSheepPosition()**, **jumpSheep()** and **gameOver()** are called. The sheep will return to its starting position (disabling the rotation commands with the mouse) and will make a small jump. This is possible by translating the sheep's body along the y axis and rotating the front legs in opposition to the rear ones. Then the end-of-game writings are loaded and a reset button will appear next to the home button.

4.3 Fly

First of all the fly should be moving together with the mouse.
This is obtained by using a raycaster object from three.js, computing the ray from the camera to the mouse, and intersecting it on a vertical plane (window plane). This intersection will determine the position of the fly.
In this way the fly will be moving around on the screen, at a distance from the frog defined by the plane z position.
This function is activated by an EventListener on the mousemove event, in this way the position of the fly is updated at every movement of the mouse.

The animation of the fly is then composed by rotations that continuously move its body, and rotations that make it flap its wings, implemented using createJS similarly as described in the above examples.

4.4 Scissors

The position of the scissors, similar to the fly, is linked to the mouse pointer making them move with it.
The animation is called through the function **animateScissor()** as follow:

```
function animateScissors(){
  createjs.Tween.get(scissorBlades[0].rotation, {loop: true})
    .to({ z: -0.1 }, 700, createjs.Ease.linear)
    .to({ z: -0.4 }, 900, createjs.Ease.linear);
  createjs.Tween.get(scissorBlades[1].rotation, {loop: true})
    .to({ z: 0.1 }, 700, createjs.Ease.linear)
    .to({ z: 0.4 }, 900, createjs.Ease.linear); }
```

It's composed of two tweens that rotate the blades (to which the handles are attached) and simulate the opening and closing of scissors.

5. User Interface

In this chapter we focus on the user interface of the project. We can see that each interaction is characterized by different backgrounds, lettering and buttons. For all of these we set a texture (image, normal, color) and we use the loadTexture(url) function that returns a Promise that resolves the TextureLoader() only when the file is loaded.

```
function loadTexture(url) {
 return new Promise(resolve => {
```

```
 new THREE.TextureLoader().load(url, resolve)
})}
```

We use this function when we have to load a texture because it allows us to avoid displaying the black geometries when the texture is not yet loaded.

5.1 Lettering

Each type of lettering is added as a texture of a BoxGeometry setting the right dimensions, and only the front face of the box contains the texture image and the three others contain a color texture that isn't visible.

5.1.1 Homepage

The title of the project is visible on the homepage, and its color depends on the selected animal. Clicking on an animal it's possible to see that there is a short description under it that is useful to help the user to better understand the aim of the game. Also in this case the chosen color for the lettering changes between frog and sheep.

5.1.2 Frog game

When the frog game starts, we can see that the background is divided into three different areas that have different dimensions and colors that are chosen with the aim to represent the different levels of danger for the fly. The largest area is yellow and is the farest from the frog and this means that when the fly is here, the frog can't reach it. To the intermediate area corresponds a more intensity of danger (orange area) than the previous, in fact here the frog tries to reach the fly but he isn't yet able. The smallest area is red and it is the most dangerous, because here the frog can catch the fly and eat it.

5.1.3 Sheep game

When the sheep game starts, we can see on the top a short title for the scene and on the bottom a brief description to rotate the sheep and shear her more easily. When the game is over it's displayed to the user the indications to play again. This was possible by adding the associated area to the scene only when the game is over, and remove it to the scene in the other cases. We set two different ways to rotate the sheep and make the shear easier: using the mouse or using the keyboard.

Mouse click

To make the sheep turn while shaving, a special function has been introduced that allows you to rotate it with a click of the mouse:

```
function rotateCameraSheepScene(){
 controls = new ObjectControls(camera, renderer.domElement,sheepBody);
 controls.setObjectToMove(sheepBody);
```

```
controls.setDistance(50, 100); // set min - max distance for zoom
controls.setZoomSpeed(0.05); // set zoom speed
controls.enableVerticalRotation();
controls.setMaxVerticalRotationAngle(Math.PI / 4, Math.PI / 4);
controls.setRotationSpeed(0.02); }
```

In this function an object of type objectControl is initialized which allows the object to rotate by clicking and dragging the mouse. In this way the view does not change but the position of the sheep does.

Arrow keys

To rotate the sheep using the keyboard we use the function rotateSheepArrows() associated with the event keydown.

```
function rotateSheepArrows(event){
    let keyCode = event.which;
    let xSpeed = 0.05;
    let ySpeed = 0.05;
    if(keyCode === 40)  // down
    {
        if(sheepBody.rotation.x <= 0.8)
        sheepBody.rotation.x += xSpeed;
    }
    if(keyCode === 39)  // right
    sheepBody.rotation.y += ySpeed;
    [...]
}
```

5.2 Buttons

Each button is created using a function which uses CircleGeometry and we give it the invisible property.

```
goButtonMaterial.transparent = true;
goButtonMaterial.opacity = 0;
```

In this way each time that we want to visualize a button we give it a visible texture depending on the selected scene.

5.2.1 Go button

In the homepage there are the "Go" buttons that we use to start the frog or the sheep game. To create the go button we use the createButton() function.

With the setButtonTexture(texturePath) function we set the texture image that depends on the selected animal. This means that we have only one button and the texture image is updated when we switch between frog and sheep.

5.2.2 Homepage button

In the frog and in the sheep game there is one button that is used back to the homepage. This button is created using the createHomeButton() function with transparent material, and then there is the setHomeButtonTexture(texturePath) function that is used to give the right texture image to the button.

5.2.3 Reset button

In the sheep game there is also the reset button. It's visible only when the user shears all the sheep's wools, because this means that the game is over and using the reset button the user can choose to start it again. The reset button is created using the createResetAnimationButton() with transparent material and then the texture image is set using the setResetAnimationButtonTexture(texturePath) function.

6. Sound Effects

6.1 Ambient sound

This section includes homepage, frog, sheep and buttons sounds. We decided to use all sounds and music in .ogg format because it is an open-source file format for multimedia and is optimized for streaming content. To implement and insert sounds into game we consulted the official documentation of three.js and proceeded as follows:

```
const homeListener = new THREE.AudioListener();
const homeAudioLoader = new THREE.AudioLoader();
homeSound = new THREE.Audio( homeListener );
homeAudioLoader.load( 'sounds/homeSound.ogg', function( buffer ) {
   homeSound.setBuffer( buffer );
   homeSound.setLoop( true );
   homeSound.setVolume( 0.5 );
   homeSound.play();
});
```

We need to use:

- AudioListener() that represents a virtual listener of the all positional and non-positional audio effects in the scene
- a global audio source with a chosen sound
- AudioLoader() to load a sound and set it as the Audio object's buffer
- setting in the AudioLoader() of some features like the volume, loop of sound, play …

For the homepage, frog scene and sheep scene we have inserted very welcoming music that changes between them but always remains in the same mood.

Starting with the ***homepage***, the music is activated by clicking for the first time on the animal you want to play with, frog or sheep.

When you enter the ***frog scene*** there will be the verses of the frog. One that is reproduced to highlight the frog's expectation to catch the fly and the other a verse of satisfaction once caught.

When you enter the ***sheep scene***, instead, the volume of music will go down once the whole sheep is sheared and she will bleat happily with the cut of her wool.

For all ***buttons*** the same sound is used, always in the gaming and chilling mood.

6.2 Interactive Sounds

6.2.1 Fly

For the fly we used the same implementation to insert and reproduce the sound but with a particularity: whenever the fly stops, its buzzing stops.
To do this we had to take the mouse and its events into account. We used: window.**addEventListener**('mousemove', **onMousePause**); to reproduce the buzz whenever the mouse is in motion. In order to stop the sound when the mouse is stopped we have inserted in the event 'mousemove' a function called each 100 millisecond that do this:

```
clearTimeout(flyTimer);
flyTimer = setTimeout(function() {
   if (currentScrren === "FROG")
   soundFly.setVolume(0);
}, 100);
```

6.2.2 Scissor

For the scissor we used the same implementation to insert and reproduce the sound but with another peculiarity. Each time the scissor passes over the sheep's body it plays its sound,

once outside the sheep's body it is stopped. To do this we always had to take into account the mouse and its events. We proceed as follows:

```
createSounds = function (event) {
   mouseSetting(event);
   for (let i = 0; i < intersects.length; i++) {
      if (intersects[i].object.id === 31) {
         sheepFlag = true;
         sheepScissorSound.setVolume(0.9);
      }
   }
   if(!sheepFlag)
      sheepScissorSound.setVolume(0);
   sheepFlag = false;
}
window.addEventListener('mousemove', createSounds);
```

We checked every time the mouse moved, if we intersect the geometry id that identifies the sheep body. If we intersect the sheep with the mouse then the volume is set to 0.9 and the sheepFlag to true. If we do not intersect, then the flag will be false and the volume will be set to 0. At the end of the for cycle the sheepFlag must be set to false to ensure that all previous checks are carried out correctly.

7. Libraries and tools

The project has been tested on the following browsers:

- Mozilla Firefox
- Google Chrome
- Microsoft Edge

To implement this project we used the following libraries:

- three.js to create all the geometries and the models
- tween.js and ease.js from the create.js library pack to animate the models
- objectcontrols to allow the rotation of the scene when needed