Final Project - Space Snake

Heidi Lohne Brække, 1886467

Space Snake

Game concept

For the final project in the course Interactive Graphics at Sapienza I created a game called Space Snake. Space snake is a simple game inspired by the old, but gold, phone snake game. The goal is to eat no that the snake grows bigger for as long as possible.

In contrast to the normal snake game, this is in 3D and a bit simplified. The snakes moves down a plane ground, in space, and needs to catch the suns to grow bigger, and avoid the moontrees. The snake is controlled by the keyboard arrows, left and right. For every sun the snake eats it grows longer, and your score increases by one. If you are good enough to reach a score of 30 the speed will increase and make it more difficult. The same happens again if you get a score higher than 50. You have 5 lives, when you hit a tree you lose a life and the ground becomes a red. For every life you lose the ground gets more red, to symbolise that you are approaching the death of the snake.

Environment

As many other web development projects this project is written mainly in javascript, with a little bit of HTML and CSS. I didn't use a specific setup for this project, I just created the files and folders needed.

The html file, which uses HTML5 only contains the links to the external libraries and the main.js file, some simple divs containing paragraphs and headers, and some CSS. All the functionality is implemented with JavaScript in the main.js file.

Libraries

Here is a short list of the external libraries I used for this project:

- three.js
- tween.js

"Three.js is a cross-browser JavaScript library that tries to make it as easy as possible to get 3D content on a webpage.Three.js is often confused with WebGL since more often than not, but not always, three.js uses WebGL to draw 3D. WebGL is a very low-level system that only draws points, lines, and triangles." Defined by Three.js Fundamentals.

“TweenJS is a simple tweening library for use in JavaScript. It was developed to integrate well with the EaselJS library, but is not dependent on or specific to it. It supports tweening of both numeric object properties & CSS style properties. The API is simple but very powerful, making it easy to create complex tweens by chaining commands.“ Defined by Tween.js.

Objects and Animation

All the objects of this game is created with the Three.js library.

Scene

The scene is where one can set up objects and where three.js is rendered. All the objects are added to the scene to make them visible. I created the scene in the init-function, as is often done since this is the base of everything.

Background

The first thing added to the scene is the background. The background is a texture created from an image of stars in space. This was implemented to kind of set the mood for the game. The background image is created as a sphere to give an illusion that the game is moving further into space.

Ground

The ground is created as a simple planeGeometry that makes it look like it is also going deeper into space. This is done by adding an x rotation of -Pi/2. Also the ground makes it easier to see the light and shadows, which I will come back to lated. I made the color on the ground transparent for the illusion of floating in space. When the ground changes color, what happens is that another ground object with a different color is placed on top of the original one. All of the other objects in the game are placed on top of the ground, so this works as a base, to align everything and make it look like it comes closer.

Snake

The snake is created from different type of cylinders in a hierarchy. The first part of the head is the parent, and then the children follow in the parts that comes after the head. All parts of the snake is referenced to as the head, because this first head element is the parent. Creating the snake as a hierarchical model made the animation of the snake smoother and easier to implement. The parent element also has the same x rotation as the ground object, where the rotation is set to pi/2 to

create the illusion that the snake is also going deeper into space. Since the snake is growing when it eats a sun, I created the snake in a for-loop.

The first to elements of the snake had to be specified in more detail than the rest of the body, as you see here. These two parts make up the head, and is what you start with at the beginning of the game. And then the body parts are added as you go.

```
function createSnakeBody(){
    var geometry;
    var material;
    var texture = new THREE.TextureLoader().load(snakeSkin);
    for (let i=0; i < 10; i+=0.15){

        if (i==0){
            geometry = new THREE.CylinderGeometry( 0.1, 0.3, 0.4, 15 );
            material = new THREE.MeshPhongMaterial( { map: texture} );
            head = new THREE.Mesh( geometry, material );
            head.position.y += 0.5;
            head.position.z += -0.5;
            head.rotation.x = Math.PI / 2;
            head.castShadow = true;
            head.receiveShadow = true;

        } else if (i==0.15){
            geometry = new THREE.CylinderGeometry( 0.3, 0.2, 0.3, 15 );
            material = new THREE.MeshPhongMaterial( { map: texture} );

            head2 = new THREE.Mesh( geometry, material );
            head2.position.y += -0.2-i;
            head2.position.z = 0;
            head2.rotation.x = 0;
            head2.castShadow = true;
            head2.receiveShadow = true;
            head.add(head2)
            scene.add(head)
        }else {
            geometry = new THREE.CylinderGeometry( 0.2, 0.2, 0.1, 15 );
            material = new THREE.MeshPhongMaterial( { map: texture} );

            bodypart = new THREE.Mesh( geometry, material );
            bodypart.position.y -= i;
            bodypart.position.z = 0;
            bodypart.rotation.x = 0;
            bodypart.castShadow = true;
            bodypart.receiveShadow = true;
            bodyparts.push(bodypart);
            head.add(bodypart)
            bodyparts.forEach(bodypart =>{
                bodypart.visible = false;
            })
        }
    }
    scene.add(head);
}
```

Every body part is the same, the only thing that changes is the distance from the head, which increases by i for every part. And all parts of the snake inherit the rotation around x, so that you get a nice 3D perspective.

As mentioned, the snake is controlled by the keyboard arrows, left and right. This is handled by an event listeners that gets activated when these keys are clicked. Further these actions send the direction of the arrow clicked on to a function that uses tween to animate the movement of the snake.

As you see here, it is a simple move, that could have been done without tween. However when using tween the animation gets smoother and more responsive, which creates a better user experience.

```
function animateSnake(direction){
    const moveAmount = 0.3
    const newXvalue = direction === 'right' ? head.position.x - moveAmount : head.position.x + moveAmount;
    const newPosition = {
        ...head.position,
        x: newXvalue,
    }
    var snakeTween = new TWEEN.Tween(head.position)
    .to(newPosition, 50)
    .start()

}
```

The material of the snake as an image, that turn into a snakeskin looking texture when added to the snake head and body.

Sun

The suns, or food elements, are created as simple sphereGeometry objects. With a for-loop, multiple suns are created in the same way, and added to a list, Foods. Also, every sun gets a random x and z position so that they are displayed all over the ground. In an effort to make it look like the snake is moving forward the sun elements are animated so that they come closer and closer by moving along the z axis. Every sun-element start at a random position in the “back”, where it can't be seen and moves towards the snake. When they move past the snake, and out of the screen they are given a new position in the back so that they can reposition and approach the snake again.

I calculated a specified distance for when an element approaching the snake collides with the snake or not, in_collision_range. So when the snake head and the sun-element is close enough, it done so that it looks like the snake eats the sun, by making it invisible, and adding a point to the score. This further leads to the illusion that a new body part is added to the snake. However, what happens is that a new body part is set to be visible.

```
foods.forEach(food =>{
    food.position.z -= speed;
    scene.add(food);
    if (food.position.z < -4){
        food.visible = true;
        food.position.z += getRandomInt(30,100);
    }
    let in_collision_range =
        Math.abs(food.position.x - head.position.x) < 0.7 &&
        Math.abs(food.position.z - head.position.z) < 0.7;
        if (in_collision_range) {
            if (food.visible){
                score += 1;
                bodyparts[score].visible = true;
                food.visible = false;
            } document.getElementById("scoreLife").innerHTML = "Score: " + score + " Lives left: "+ life;
        } else {
            score = score
            document.getElementById("scoreLife").innerHTML = "Score: " + score + " Lives left: "+ life;
        }
    }
)
```

Moontree

The moontree is created as a simple hierarchy, the tree trunk is the parent and then the moons that make up the top of the tree, are children. This was inspired by the second homework, I just made it a bit more outer space appropriate. The animation

of the moontree is similar to the suns, only when within the collision range, the snake loses a life.

Light

I have implemented both a pointlight and ambient light for this project. The point light is positioned and makes the elements cast shadows on each other to make it look more realistic. Further the ambient light does not create shadows and does not have directions in the same way. However this makes the texture and shape of the objects more visible and realistic.

Other Technical details

In addition to the user interaction that lets you control the snake, the game is started, and restarted, with a button click and when the space key is pressed. When the game ends, because you have lost all the 5 lives you were given, pressing the space button reloads the entire page. This is not the best way to do this, but I think it works for a small game like this. If it had been a bigger and more complex game this action would have been to expensive.

The instructions and game over info are simple div tags that are hidden and displayed when it is natural. The state of the game is controlled by a boolean variable, gameOver, so that the system “knows” when to display what.

Further the game is responsive to the window size, by updating the perspective matrix for the camera when the window changes.

Source:

https://threejsfundamentals.org/threejs/lessons/threejs-fundamentals.html
https://createjs.com/tweenjs