SAPIENZA
UNIVERSITÀ DI ROMA

Interactive graphics

Final project report

Snake game

Instructor:

Professor Schaerf Marco

Groups Member:

Ahmadreza Fardaei-1905981

Bamdad Kianmehr-1918176

26 July, 2020

Technical presentation

The purpose of this project is developing a game called SNAKE (or sometimes worm) and is famous because of using in various mobile phones. The first version has been designed in the year 1976 and was used as a first release in ATARI in the year 1977. It is a snake that goes around the specific environment to collect a goal (apple). Collecting goals (apples) will give score and increase its body length while game over is the result of having an accident with its body.

The game developed in this project is like the simple classic snake game, designing a cube in the whole environment, define the world of snake to move inside.

I. Description of the environment and libraries used in the project

Below we can see the libraries which were used in this project:

- **TreeJs**: this is a library and application programming interface (API) used to create and display animated 3D computer graphics in a web browser.

- **Dat.GUI**: is a lightweight graphical user interface (GUI) for changing variables in JavaScript.

- **OrbitControls:** allows the camera to orbit around a target

II. Description of all the technical aspects of the project

TreeJs is used to render the scene, in order to setup the dynamic world that would represent the snake and goal (apple) in the game scene. Steps to implement the game are described as follows:

- **Dynamic world:**

  Our dynamic world is containing the snake and the goal. This part is responsible to create a goal, a new snake and increase the length of snake when it reaches at a goal. Some samples of codes are shown below:

```
for(var i = snake.length - 2; i > -1; i--){
    if(head.mesh.position.distanceTo(snake[i].mesh.position) < 1){
         end = true;
         break;
       }
  }
                if(end) {
                    restart();
                }
                if(head.mesh.position.distanceTo(apple.mesh.position) < 1){
                    apple.setPosition(spawnAppleVector());
                    text.innerHTML = "Score: " + (++score);
                    snake.unshift(new Cube( new THREE.Vector3(snake[0].mesh.posit
ion.x, snake[0].mesh.position.y, snake[0].mesh.position.z), new THREE.MeshPhongMa
terial( { color: 0x388e3c} ), scene));
```

- **Scene render:**

  In order to render the game scene with all its components we need to rendering a scene which has this responsibility to create all the components geometry, assigns a texture to components, components mesh creates a ThreeJs Scene, a ThreeJs and put them at a pre-arranged location in the rendered game scene. Some samples of codes are shown below:

```
var renderer = new THREE.WebGLRenderer(), camera = new THREE.PerspectiveCamera(45
, aspectRatio, 0.1, 1000), scene = new THREE.Scene();
scene.background = new THREE.Color( BACKGROUND_COLOR );

//adding light:
scene.add(light)

//adding mesh:
scene.add(this.mesh);
```

- **Updating the dynamic world:**

  When the snake eats the apple, we need to change the location of the apple in the environment in other hand the snake is moving through the space and need to be update for each motion:

  Our update sample codes when we want to change the direction of movement of the snake:

```
    function handleKeyDown( event ) {

        //console.log( 'handleKeyDown' );

        switch ( event.keyCode ) {
            case scope.keys.UP:
                pan( 0, scope.keyPanSpeed );
                scope.update();
                break;

            case scope.keys.BOTTOM:
                pan( 0, - scope.keyPanSpeed );
                scope.update();
                break;

            case scope.keys.LEFT:
                pan( scope.keyPanSpeed, 0 );
                scope.update();
                break;

            case scope.keys.RIGHT:
                pan( - scope.keyPanSpeed, 0 );
                scope.update();
                break;

        }
```

In addition, in case that head of the snake reaches the apple, score will be added by one and length of the snake will be increased:

```
if(head.mesh.position.distanceTo(apple.mesh.position) < 1){
                    apple.setPosition(spawnAppleVector());
                    text.innerHTML = "Score: " + (++score);

                    snake.unshift(new Cube( new THREE.Vector3(snake[0].mesh.posit
ion.x, snake[0].mesh.position.y, snake[0].mesh.position.z), new THREE.MeshPhongMa
terial( { color: 0x388e3c} ), scene));

                }
```

In case that head of the snake touches its body, game will be finished and restarted again:

```
                for(var i = snake.length - 2; i > -1; i--){
                    if(head.mesh.position.distanceTo(snake[i].mesh.position) < 1)
{
```

```
                        end = true;
                        break;
                    }
                }

                if(end) {
                    restart();
                }
```

- **Game state animation:**
  After defining the head and the tail(body) and their color, we represent the change of position and direction where it is changed by turning the head and after changing, tail must follow the head again:

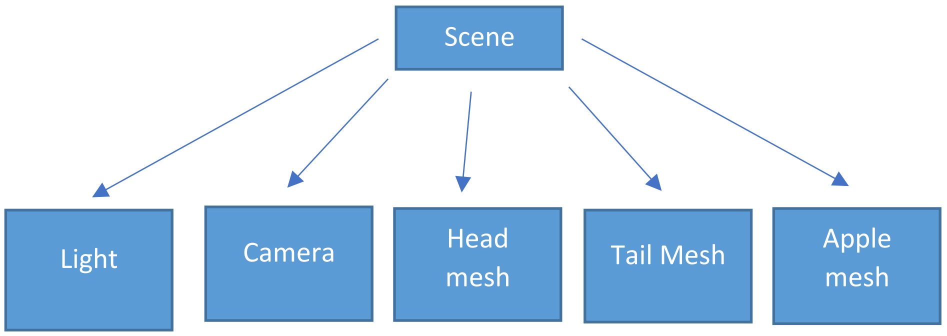
```
requestAnimationFrame(render);

            tetha += clock.getDelta();

            if(tetha > delta){
                var tail = snake.shift();
                var head = snake[snake.length - 1];

                head.mesh.material.color.setHex(BODY_COLOR);
                tail.mesh.material.color.setHex(HEAD_COLOR);

                direction = keysQueue.length > 0 ? keysQueue.pop(0) : direction;
                var newPosition = new THREE.Vector3(head.mesh.position.x + direct
ion.x + Math.sign(direction.x) * padding, head.mesh.position.y + direction.y + Ma
th.sign(direction.y) * padding, head.mesh.position.z + direction.z + Math.sign(di
rection.z) * padding);
                tail.setPosition(newPosition);

                snake.push(tail);
                head = tail;
```

III. Description of the implemented interactions

- **Hierarchical Models**

In order to create a hierarchical model of the bounce game which is composed of the following parts:

- ✓ **Light**
- ✓ **Camera**
- ✓ **Head mesh**
- ✓ **Tail mesh**
- ✓ **Apple mesh**



The game is modeled by using hierarchical modeling (in the above figure). All these components are added to scene in a hierarchical way. To add parts of these components ThreeJs geometry is used.

- **Lights and texture:**

Since the game field required to be obvious, we could not add a texture to it, and for snake and the apple it was better to be colored in order to show the effect of lights in a better way, so we just added a texture to scene background, using three.js library for uploading the image:

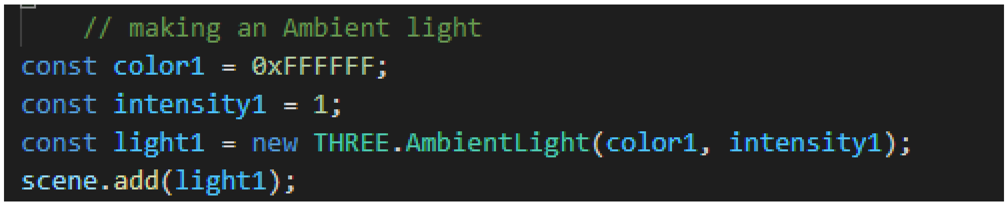
```
        // texture bakground:
        const loader = new THREE.TextureLoader();
        const bgTexture = loader.load('https://threejsfundamentals.org/threejs/resources/images/daikanyam
a.jpg');
        scene.background = bgTexture;
```

for the lights, we wanted to see the effect of directional light, ambient light and hemispherical light.

To adjust the light's parameters, we'll use dat.GUI library.

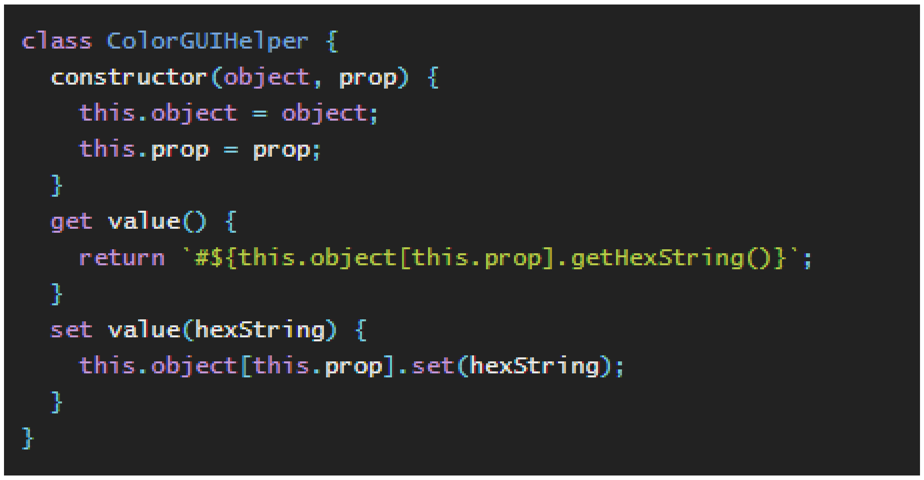
We'll make a helper which looks like a CSS hex color string (e.i #FF8844), which gets the color from a named property, convert it to a hex string to offer to dat.GUI.

1- First, we have to import GUI library
2- Then create the Ambient light:



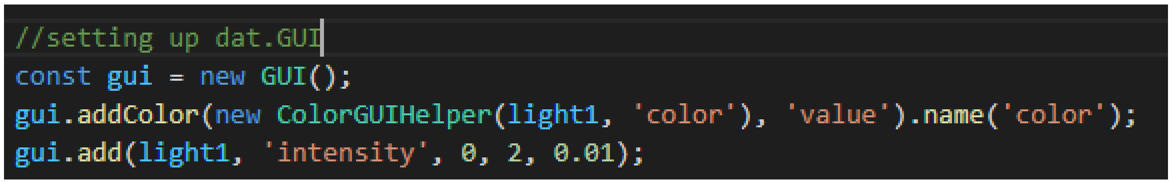
```
    // making an Ambient light
const color1 = 0xFFFFFF;
const intensity1 = 1;
const light1 = new THREE.AmbientLight(color1, intensity1);
scene.add(light1);
```

3- This is the helper:



```
class ColorGUIHelper {
  constructor(object, prop) {
    this.object = object;
    this.prop = prop;
  }
  get value() {
    return `#${this.object[this.prop].getHexString()}`;
  }
  set value(hexString) {
    this.object[this.prop].set(hexString);
  }
}
```

4- Setting up the dat.GUI:



```
//setting up dat.GUI
const gui = new GUI();
gui.addColor(new ColorGUIHelper(light1, 'color'), 'value').name('color');
gui.add(light1, 'intensity', 0, 2, 0.01);
```

** The Ambient-light effectively just multiplies the material's color by the light's color times the intensity.

- **Hemisphere-Light:**

A Hemisphere-light takes a sky color and a ground color and just multiplies the material's color between those 2 colors—the sky color if the surface of the object is pointing up and the ground color if the surface of the object is pointing down.

```
// Hemisphere light
const skyColor = 0xB1E1FF;  // light blue
const groundColor = 0xB97A20;  // brownish orange
const intensity2 = 1;
const light2 = new THREE.HemisphereLight(skyColor, groundColor, intensity2);
scene.add(light2);
```

```
// Hemisphere:
const gui2 = new dat.GUI();
gui2.addColor(new ColorGUIHelper(light2, 'color'), 'value').name('skyColor');
gui2.addColor(new ColorGUIHelper(light2, 'groundColor'), 'value').name('groundColor');
gui2.add(light2, 'intensity', 0, 15, 0.01);
```

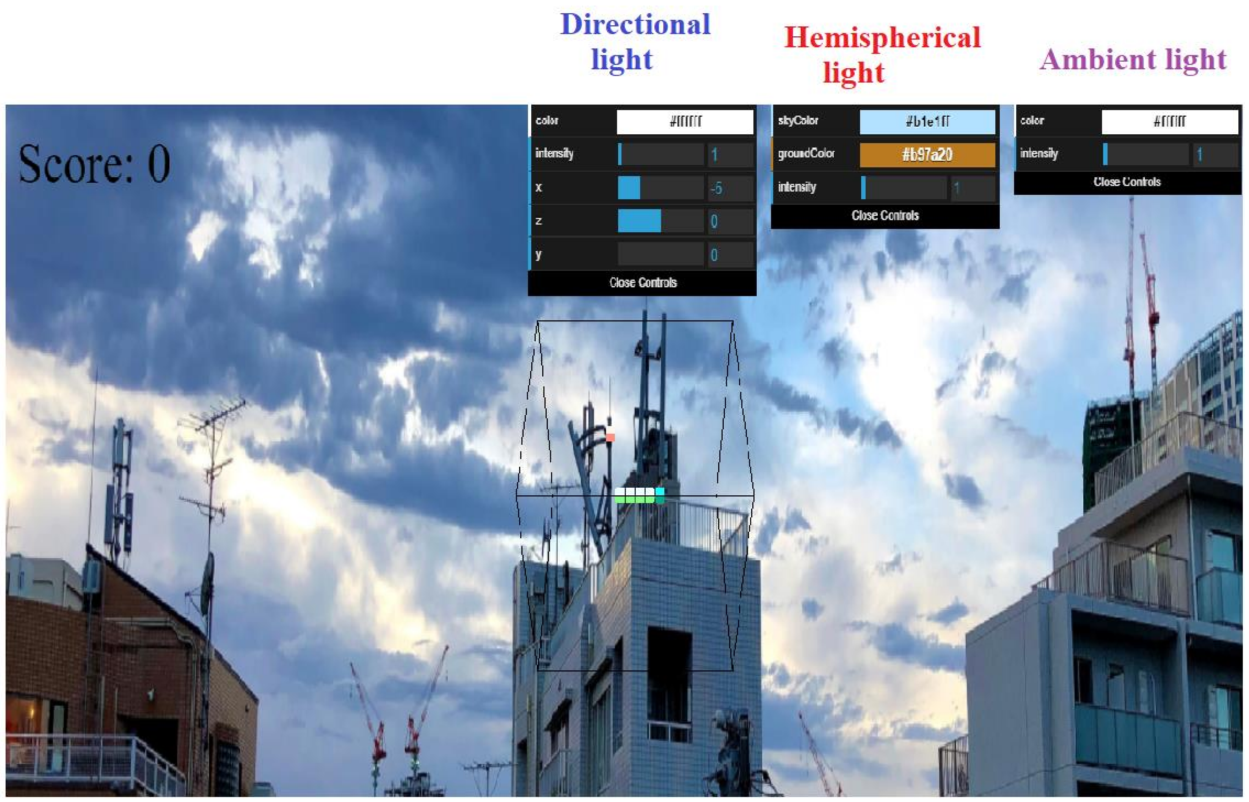
- **Directional-Light:**

A Directional-Light is often used to represent the sun.

```
// Directional light
const color3 = 0xFFFFFF;
const intensity3 = 1;
const light3 = new THREE.DirectionalLight(color3, intensity3);
light3.position.set(0, 10, 0);
light3.target.position.set(-5, 0, 0);
scene.add(light3);
scene.add(light3.target);
```

```
// Directional:
const gui3 = new dat.GUI();
gui3.addColor(new ColorGUIHelper(light3, 'color'), 'value').name('color');
gui3.add(light3, 'intensity', 0, 25, 0.01);
gui3.add(light3.target.position, 'x', -10, 10);
gui3.add(light3.target.position, 'z', -10, 10);
gui3.add(light3.target.position, 'y', 0, 10);
```

And this is the final result:



color #ffffff
intensity 1
x -5
z 0
y 0

skyColor #b1e1ff
groundColor #b97a20
intensity 1

color #ffffff
intensity 1

How to play?

to play this game users don't need to install it or do something strange just they need a web browser like Google Chrome, Mozilla Fire-Fox, Microsoft Edge and all other browsers which support webgl. Then they will open the HTML file on the browser.

As soon as the game opened up on browser, user will see a cube in the center of the page which is the environment of the snake.

The Snake will start to move automatically and the user just have to control its direction to reach the goal (Apple). Using mouse could change the point of view to see the environment in a way which user is more comfortable with.

The keyboard keys that will be used to play this game and control the snake movement are:

"Q" to move Up (Z direction in cartesian space)

"A" to move Down (-Z direction in cartesian space)

"Arrow Key Up" to move in depth in (X direction in cartesian space)

“Arrow Key Down” to move in depth out (-X direction in cartesian space)

“Arrow Key Right” to move right (Y direction in cartesian space)

“Arrow Key Left” to move left (-Y direction in cartesian space)

Above mentioned keys, depends on the point of view that the user will choose could be different.

References:

https://threejs.org/

https://github.com/dataarts/dat.gui