COMP 370 Individual Project Part 2: Final Version

Thomas Williamson

id: 588206

2021/12/07

Abstract:

Interactive webgame the goal of the game is dodge the incoming blocks of “space debris” the high score and score are present at the top of the screen, there where little to no changes in from the original

Setup:

In order to load you must access the assignment7.html through a server, suggested application serverz

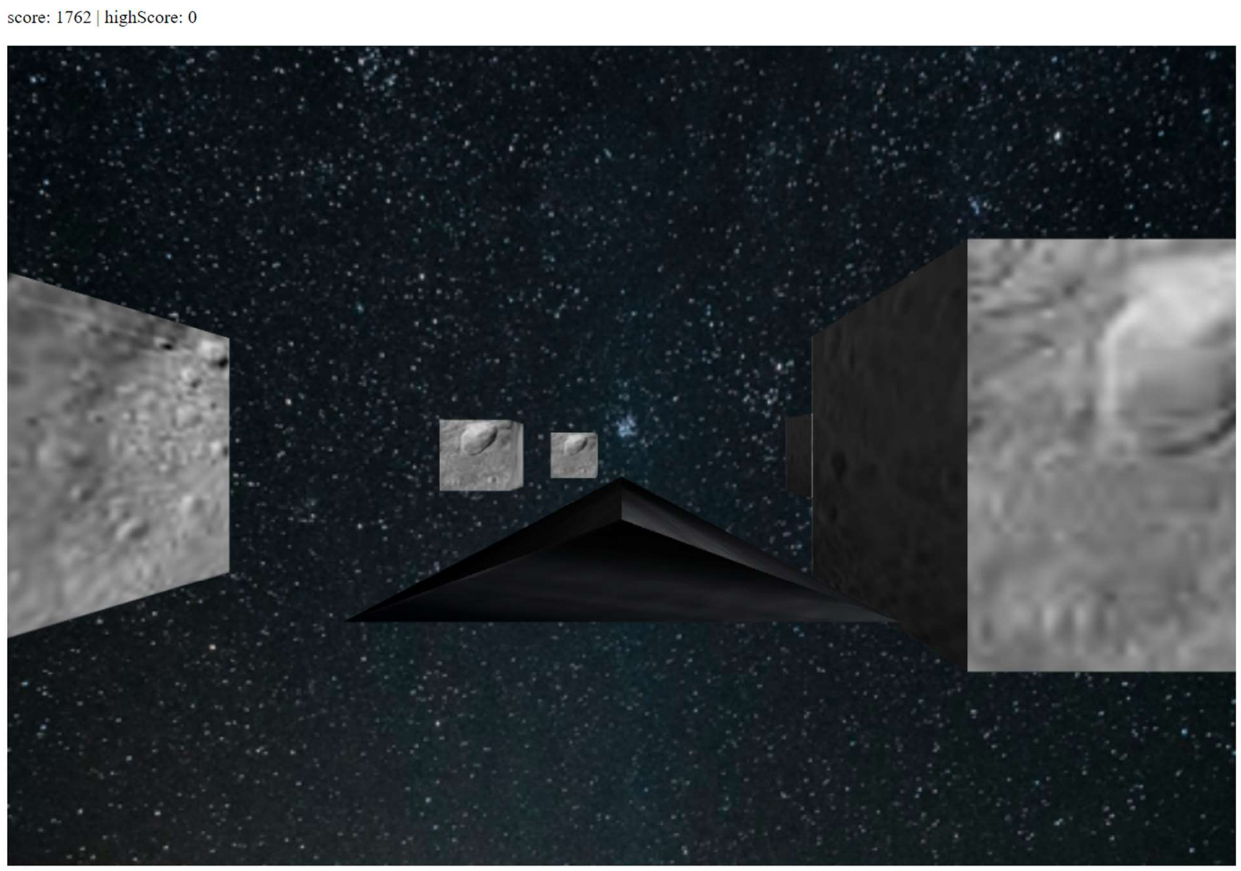
Controls:

to move left and right either use A and D or the left and right arrow keys, to speed up and slow down press W and S or the up and down arrow keys.

Take away:

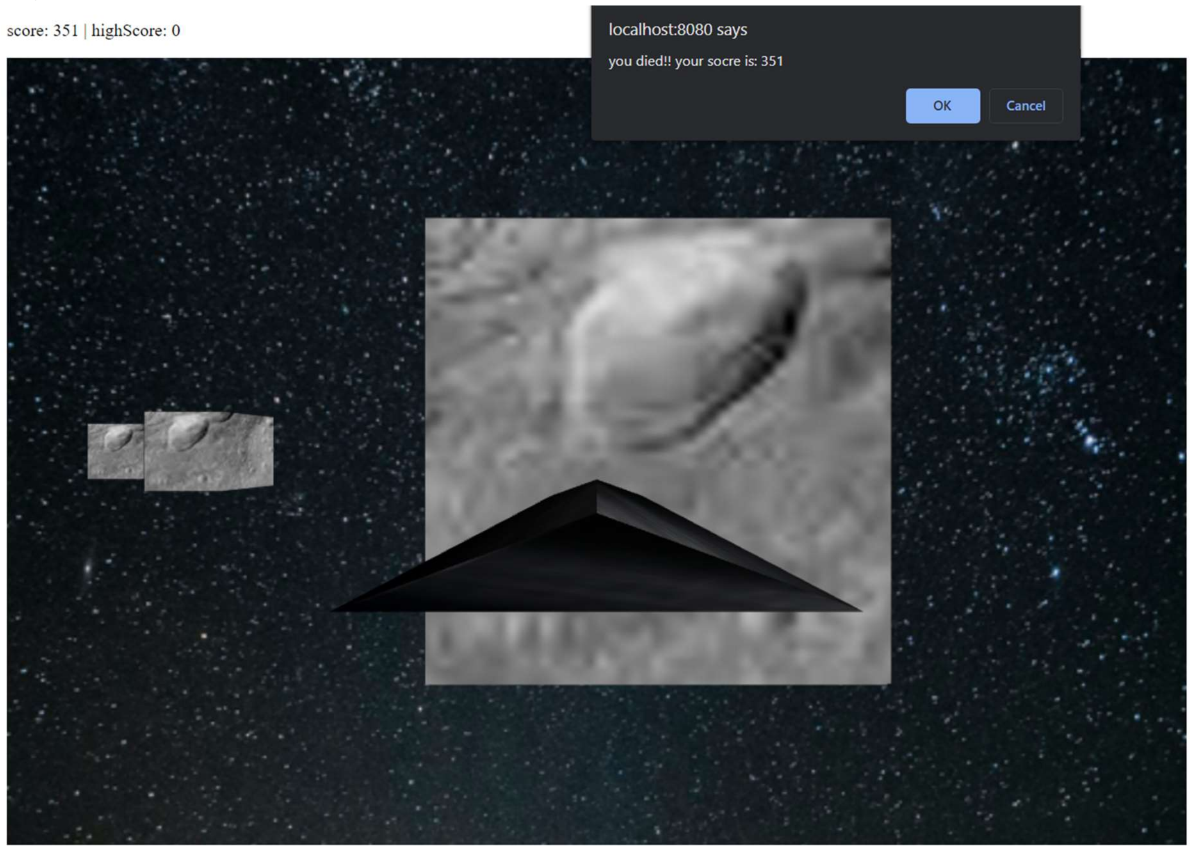
It was quite a lot easier than I anticipated. In the process of the development, I learned how to implement multiple different shapes I made use of instancing as well as unseen surfaces in order to put the background.

Gameplay:



score: 1762 | highScore: 0

impact:

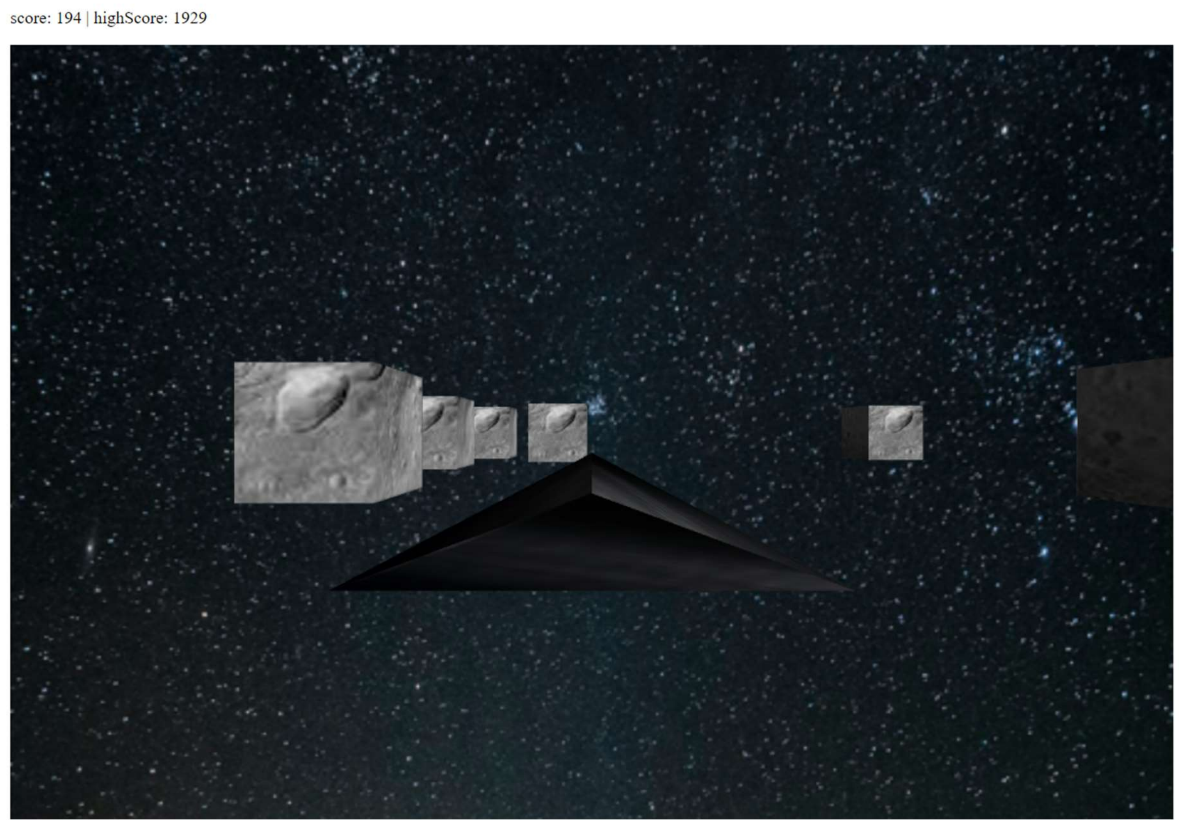


score: 351 | highScore: 0

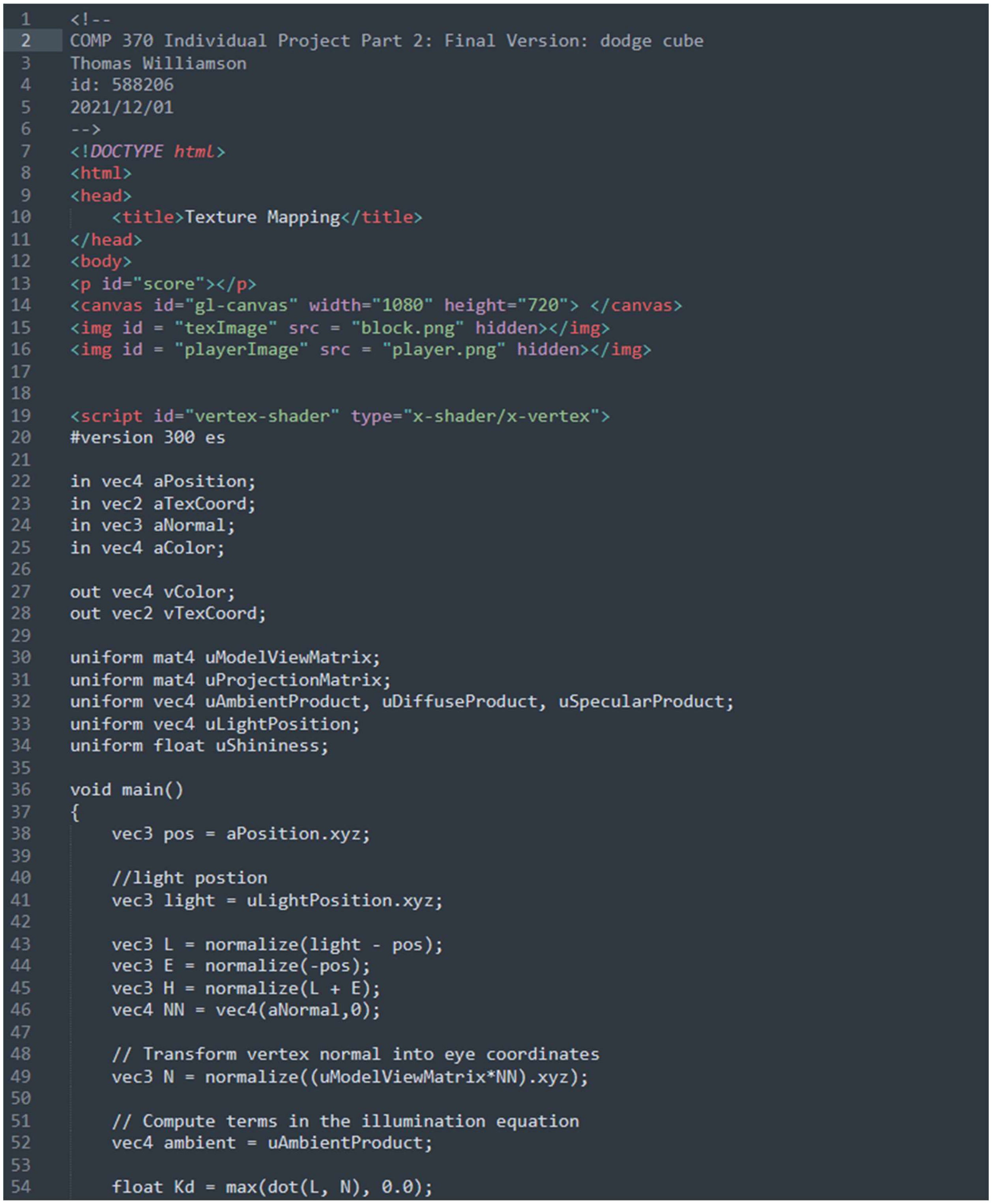
localhost:8080 says

you died!! your socre is: 351

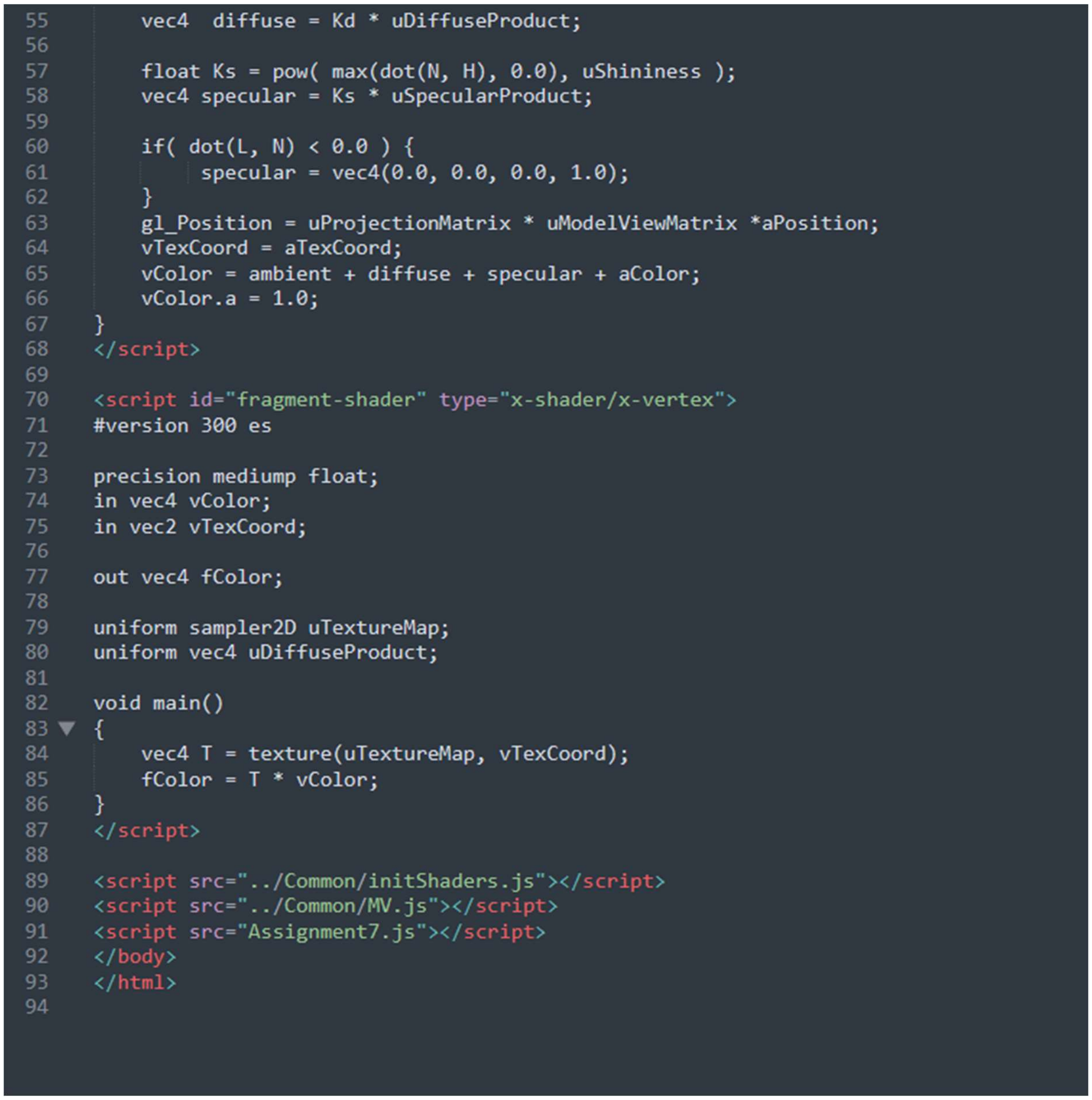
high Score


score: 194 | highScore: 1929

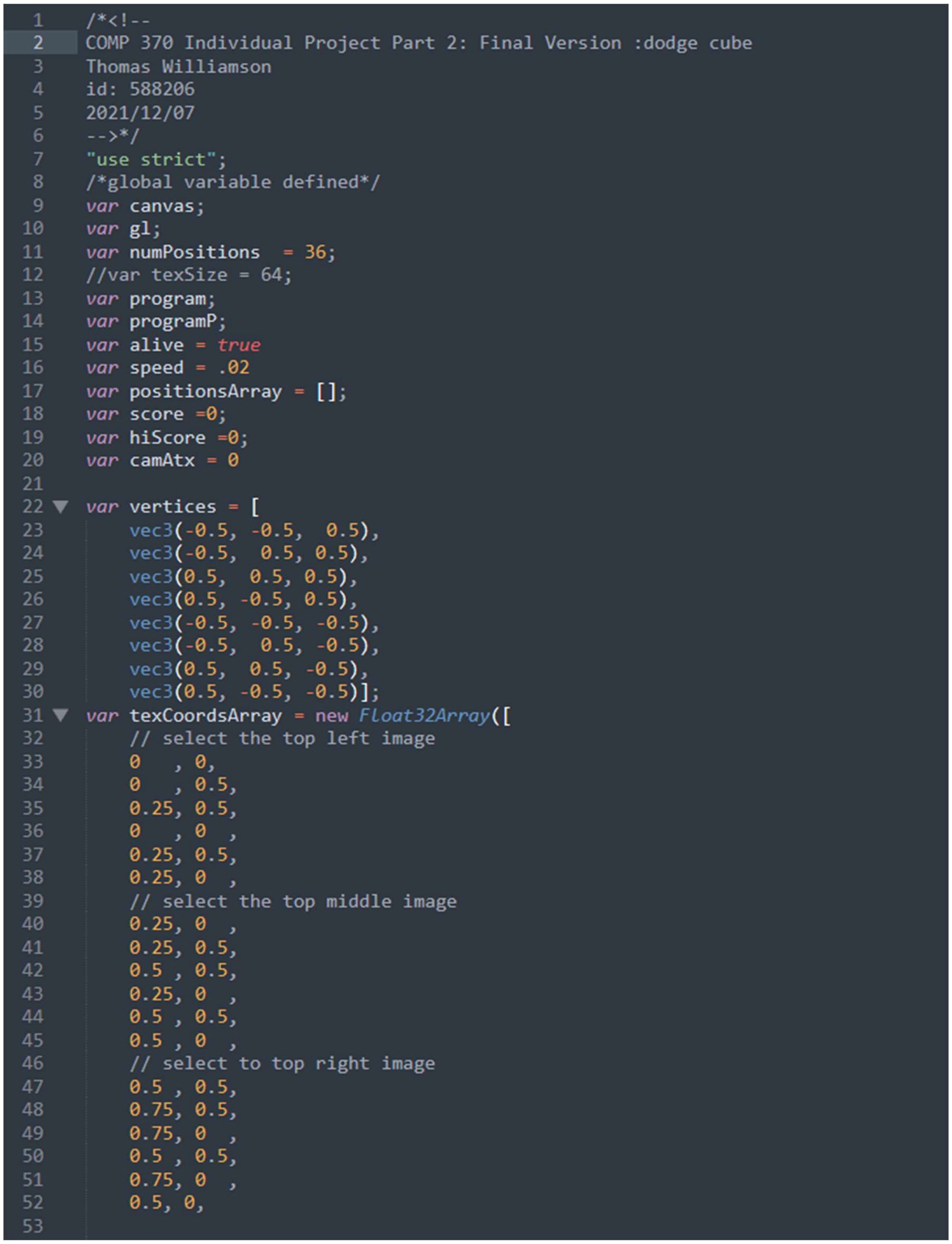
code:



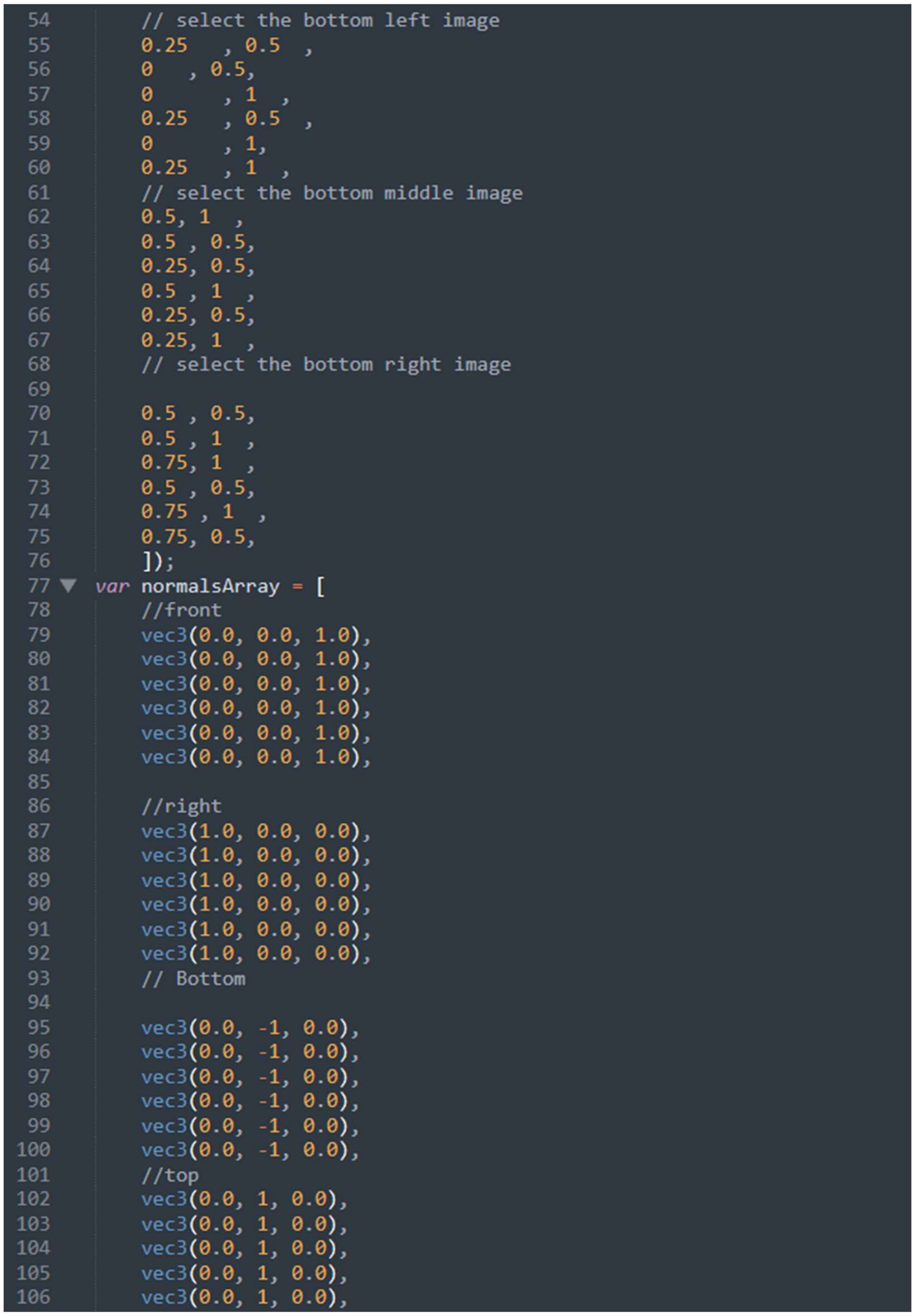
```
<!--
COMP 370 Individual Project Part 2: Final Version: dodge cube
Thomas Williamson
id: 588206
2021/12/01
-->
<!DOCTYPE html>
<html>
<head>
    <title>Texture Mapping</title>
</head>
<body>
<p id="score"></p>
<canvas id="gl-canvas" width="1080" height="720"> </canvas>
<img id = "texImage" src = "block.png" hidden></img>
<img id = "playerImage" src = "player.png" hidden></img>


<script id="vertex-shader" type="x-shader/x-vertex">
#version 300 es

in vec4 aPosition;
in vec2 aTexCoord;
in vec3 aNormal;
in vec4 aColor;

out vec4 vColor;
out vec2 vTexCoord;

uniform mat4 uModelViewMatrix;
uniform mat4 uProjectionMatrix;
uniform vec4 uAmbientProduct, uDiffuseProduct, uSpecularProduct;
uniform vec4 uLightPosition;
uniform float uShininess;

void main()
{
    vec3 pos = aPosition.xyz;

    //light postion
    vec3 light = uLightPosition.xyz;

    vec3 L = normalize(light - pos);
    vec3 E = normalize(-pos);
    vec3 H = normalize(L + E);
    vec4 NN = vec4(aNormal,0);

    // Transform vertex normal into eye coordinates
    vec3 N = normalize((uModelViewMatrix*NN).xyz);

    // Compute terms in the illumination equation
    vec4 ambient = uAmbientProduct;

    float Kd = max(dot(L, N), 0.0);
```



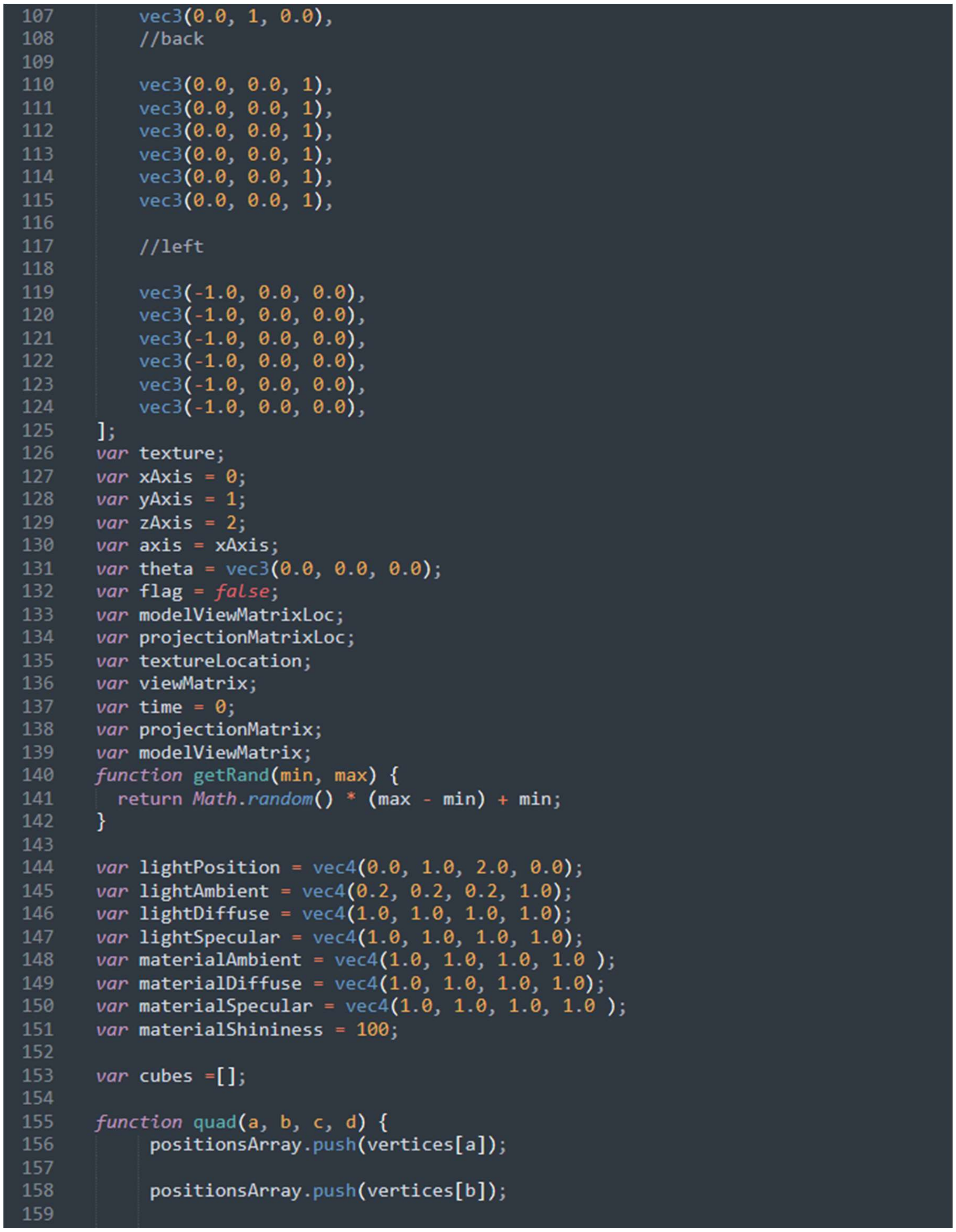
```
    vec4  diffuse = Kd * uDiffuseProduct;

    float Ks = pow( max(dot(N, H), 0.0), uShininess );
    vec4 specular = Ks * uSpecularProduct;

    if( dot(L, N) < 0.0 ) {
        specular = vec4(0.0, 0.0, 0.0, 1.0);
    }
    gl_Position = uProjectionMatrix * uModelViewMatrix *aPosition;
    vTexCoord = aTexCoord;
    vColor = ambient + diffuse + specular + aColor;
    vColor.a = 1.0;
}
</script>

<script id="fragment-shader" type="x-shader/x-vertex">
#version 300 es

precision mediump float;
in vec4 vColor;
in vec2 vTexCoord;

out vec4 fColor;

uniform sampler2D uTextureMap;
uniform vec4 uDiffuseProduct;

void main()
{
    vec4 T = texture(uTextureMap, vTexCoord);
    fColor = T * vColor;
}
</script>

<script src="../Common/initShaders.js"></script>
<script src="../Common/MV.js"></script>
<script src="Assignment7.js"></script>
</body>
</html>
```



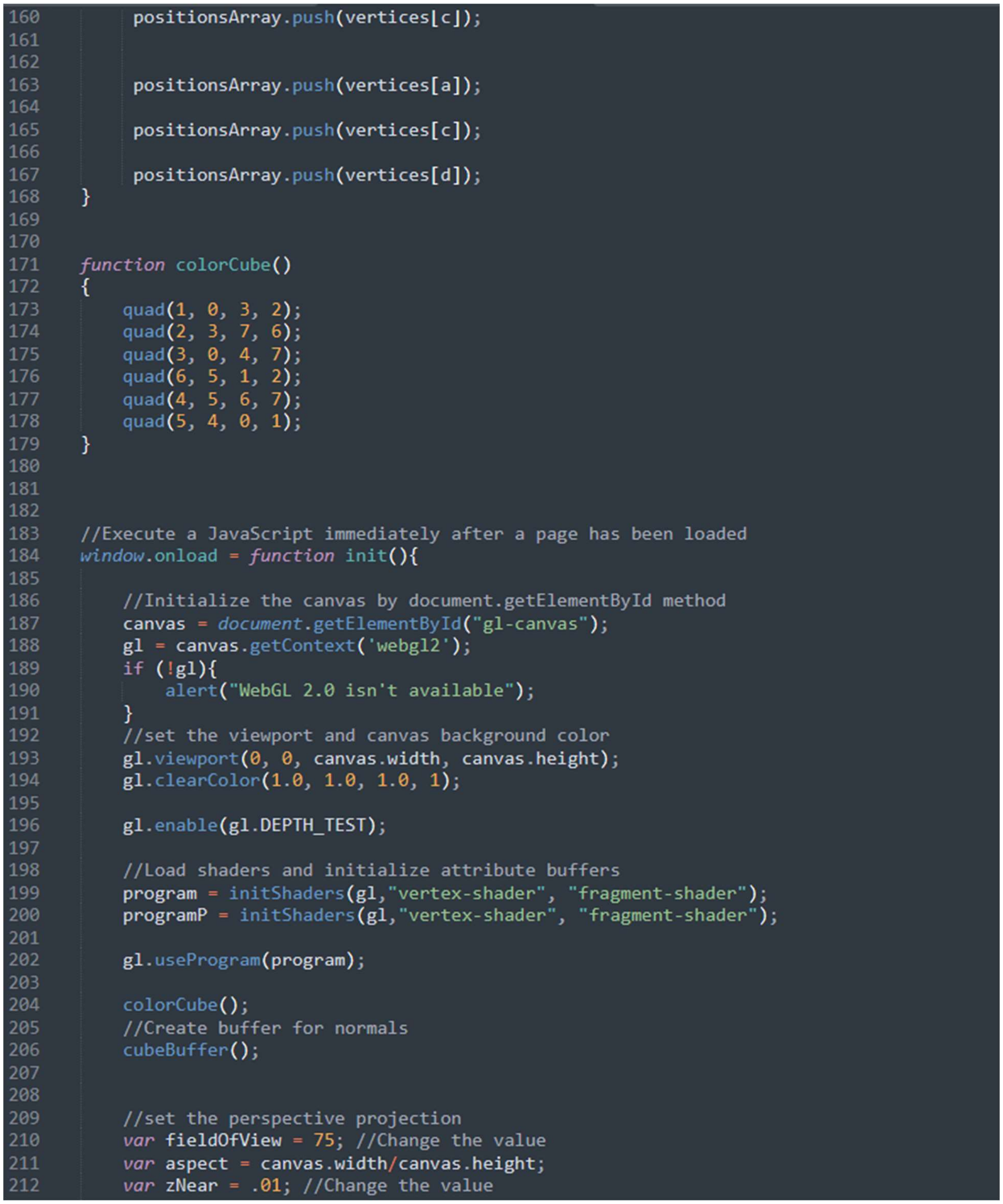
```
/*<!--
COMP 370 Individual Project Part 2: Final Version :dodge cube
Thomas Williamson
id: 588206
2021/12/07
-->*/
"use strict";
/*global variable defined*/
var canvas;
var gl;
var numPositions  = 36;
//var texSize = 64;
var program;
var programP;
var alive = true
var speed = .02
var positionsArray = [];
var score =0;
var hiScore =0;
var camAtx = 0

var vertices = [
    vec3(-0.5, -0.5,  0.5),
    vec3(-0.5,  0.5, 0.5),
    vec3(0.5,  0.5, 0.5),
    vec3(0.5, -0.5, 0.5),
    vec3(-0.5, -0.5, -0.5),
    vec3(-0.5,  0.5, -0.5),
    vec3(0.5,  0.5, -0.5),
    vec3(0.5, -0.5, -0.5)];
var texCoordsArray = new Float32Array([
    // select the top left image
    0   , 0,
    0   , 0.5,
    0.25, 0.5,
    0   , 0  ,
    0.25, 0.5,
    0.25, 0  ,
    // select the top middle image
    0.25, 0  ,
    0.25, 0.5,
    0.5 , 0.5,
    0.25, 0  ,
    0.5 , 0.5,
    0.5 , 0  ,
    // select to top right image
    0.5 , 0.5,
    0.75, 0.5,
    0.75, 0  ,
    0.5 , 0.5,
    0.75, 0  ,
    0.5, 0,

```



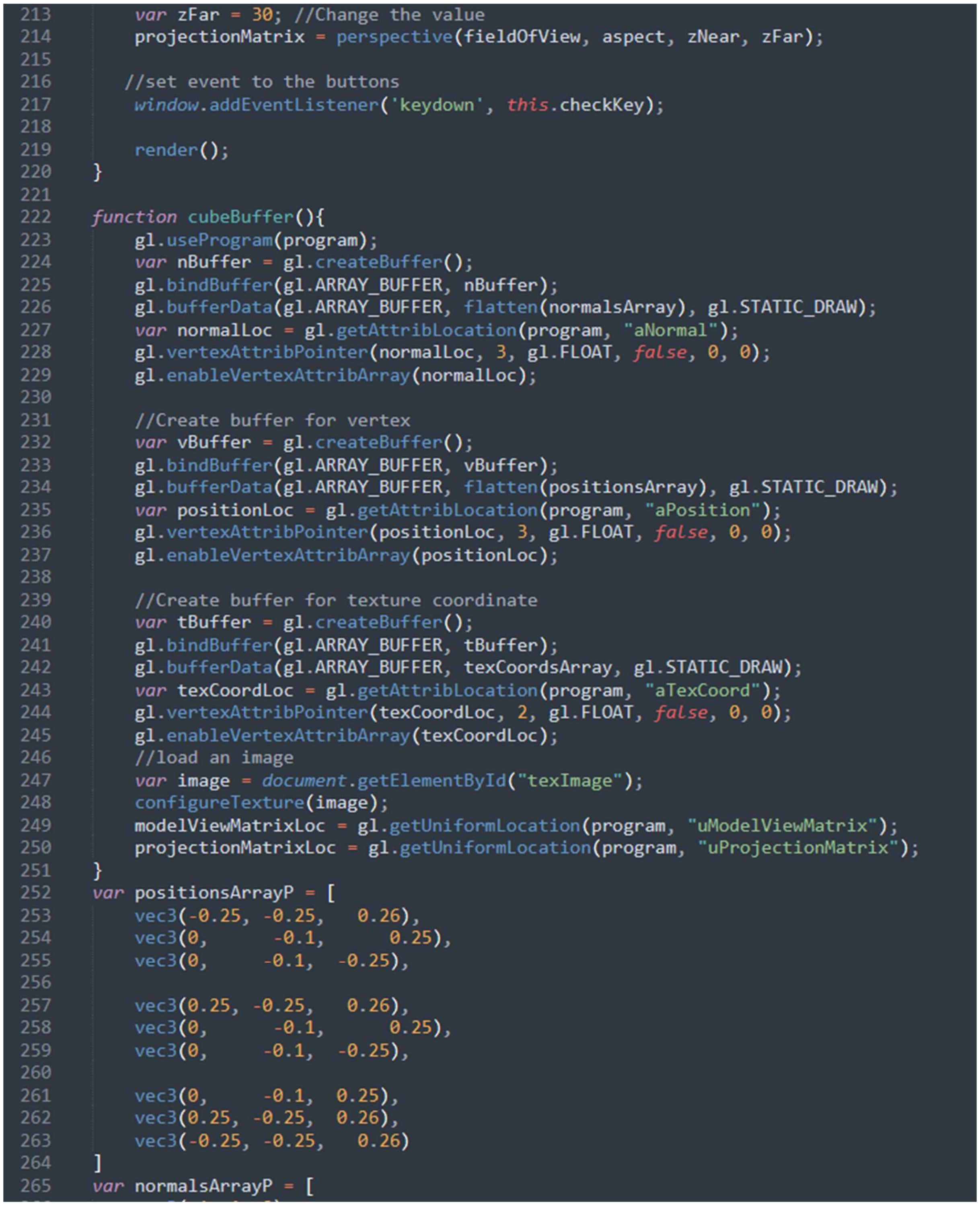
```
    // select the bottom left image
    0.25    , 0.5   ,
    0    , 0.5,
    0       , 1   ,
    0.25    , 0.5   ,
    0       , 1,
    0.25    , 1   ,
    // select the bottom middle image
    0.5, 1   ,
    0.5 , 0.5,
    0.25, 0.5,
    0.5 , 1   ,
    0.25, 0.5,
    0.25, 1   ,
    // select the bottom right image

    0.5 , 0.5,
    0.5 , 1   ,
    0.75, 1   ,
    0.5 , 0.5,
    0.75 , 1   ,
    0.75, 0.5,
    ]);
var normalsArray = [
    //front
    vec3(0.0, 0.0, 1.0),
    vec3(0.0, 0.0, 1.0),
    vec3(0.0, 0.0, 1.0),
    vec3(0.0, 0.0, 1.0),
    vec3(0.0, 0.0, 1.0),
    vec3(0.0, 0.0, 1.0),

    //right
    vec3(1.0, 0.0, 0.0),
    vec3(1.0, 0.0, 0.0),
    vec3(1.0, 0.0, 0.0),
    vec3(1.0, 0.0, 0.0),
    vec3(1.0, 0.0, 0.0),
    vec3(1.0, 0.0, 0.0),
    // Bottom

    vec3(0.0, -1, 0.0),
    vec3(0.0, -1, 0.0),
    vec3(0.0, -1, 0.0),
    vec3(0.0, -1, 0.0),
    vec3(0.0, -1, 0.0),
    vec3(0.0, -1, 0.0),
    //top
    vec3(0.0, 1, 0.0),
    vec3(0.0, 1, 0.0),
    vec3(0.0, 1, 0.0),
    vec3(0.0, 1, 0.0),
    vec3(0.0, 1, 0.0),
```



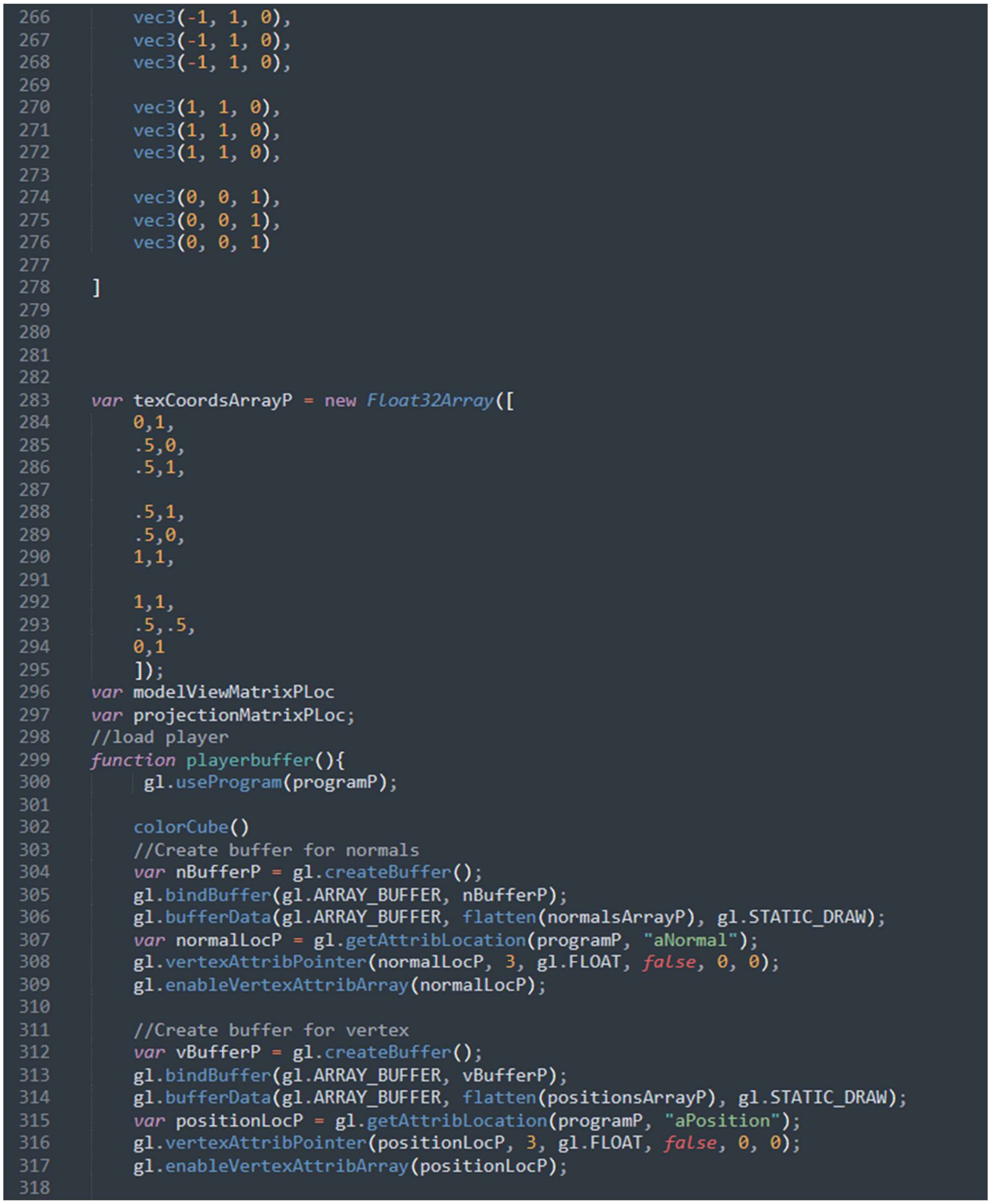
```
        vec3(0.0, 1, 0.0),
        //back

        vec3(0.0, 0.0, 1),
        vec3(0.0, 0.0, 1),
        vec3(0.0, 0.0, 1),
        vec3(0.0, 0.0, 1),
        vec3(0.0, 0.0, 1),
        vec3(0.0, 0.0, 1),

        //left

        vec3(-1.0, 0.0, 0.0),
        vec3(-1.0, 0.0, 0.0),
        vec3(-1.0, 0.0, 0.0),
        vec3(-1.0, 0.0, 0.0),
        vec3(-1.0, 0.0, 0.0),
        vec3(-1.0, 0.0, 0.0),
    ];
    var texture;
    var xAxis = 0;
    var yAxis = 1;
    var zAxis = 2;
    var axis = xAxis;
    var theta = vec3(0.0, 0.0, 0.0);
    var flag = false;
    var modelViewMatrixLoc;
    var projectionMatrixLoc;
    var textureLocation;
    var viewMatrix;
    var time = 0;
    var projectionMatrix;
    var modelViewMatrix;
    function getRand(min, max) {
      return Math.random() * (max - min) + min;
    }

    var lightPosition = vec4(0.0, 1.0, 2.0, 0.0);
    var lightAmbient = vec4(0.2, 0.2, 0.2, 1.0);
    var lightDiffuse = vec4(1.0, 1.0, 1.0, 1.0);
    var lightSpecular = vec4(1.0, 1.0, 1.0, 1.0);
    var materialAmbient = vec4(1.0, 1.0, 1.0, 1.0 );
    var materialDiffuse = vec4(1.0, 1.0, 1.0, 1.0);
    var materialSpecular = vec4(1.0, 1.0, 1.0, 1.0 );
    var materialShininess = 100;

    var cubes =[];

    function quad(a, b, c, d) {
         positionsArray.push(vertices[a]);

         positionsArray.push(vertices[b]);

```



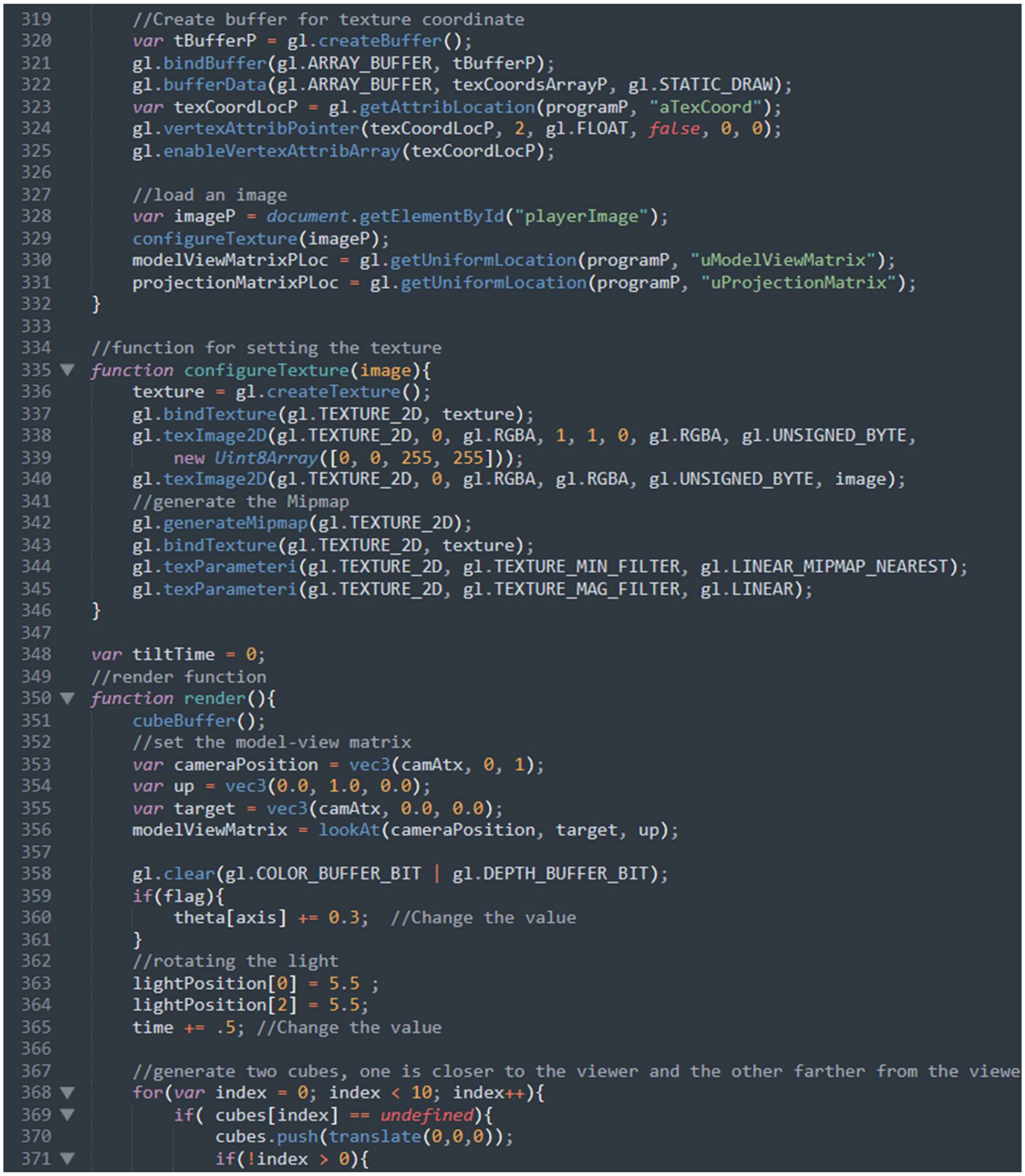
```
        positionsArray.push(vertices[c]);


        positionsArray.push(vertices[a]);

        positionsArray.push(vertices[c]);

        positionsArray.push(vertices[d]);
}


function colorCube()
{
    quad(1, 0, 3, 2);
    quad(2, 3, 7, 6);
    quad(3, 0, 4, 7);
    quad(6, 5, 1, 2);
    quad(4, 5, 6, 7);
    quad(5, 4, 0, 1);
}



//Execute a JavaScript immediately after a page has been loaded
window.onload = function init(){

    //Initialize the canvas by document.getElementById method
    canvas = document.getElementById("gl-canvas");
    gl = canvas.getContext('webgl2');
    if (!gl){
        alert("WebGL 2.0 isn't available");
    }
    //set the viewport and canvas background color
    gl.viewport(0, 0, canvas.width, canvas.height);
    gl.clearColor(1.0, 1.0, 1.0, 1);

    gl.enable(gl.DEPTH_TEST);

    //Load shaders and initialize attribute buffers
    program = initShaders(gl,"vertex-shader", "fragment-shader");
    programP = initShaders(gl,"vertex-shader", "fragment-shader");

    gl.useProgram(program);

    colorCube();
    //Create buffer for normals
    cubeBuffer();


    //set the perspective projection
    var fieldOfView = 75; //Change the value
    var aspect = canvas.width/canvas.height;
    var zNear = .01; //Change the value
```



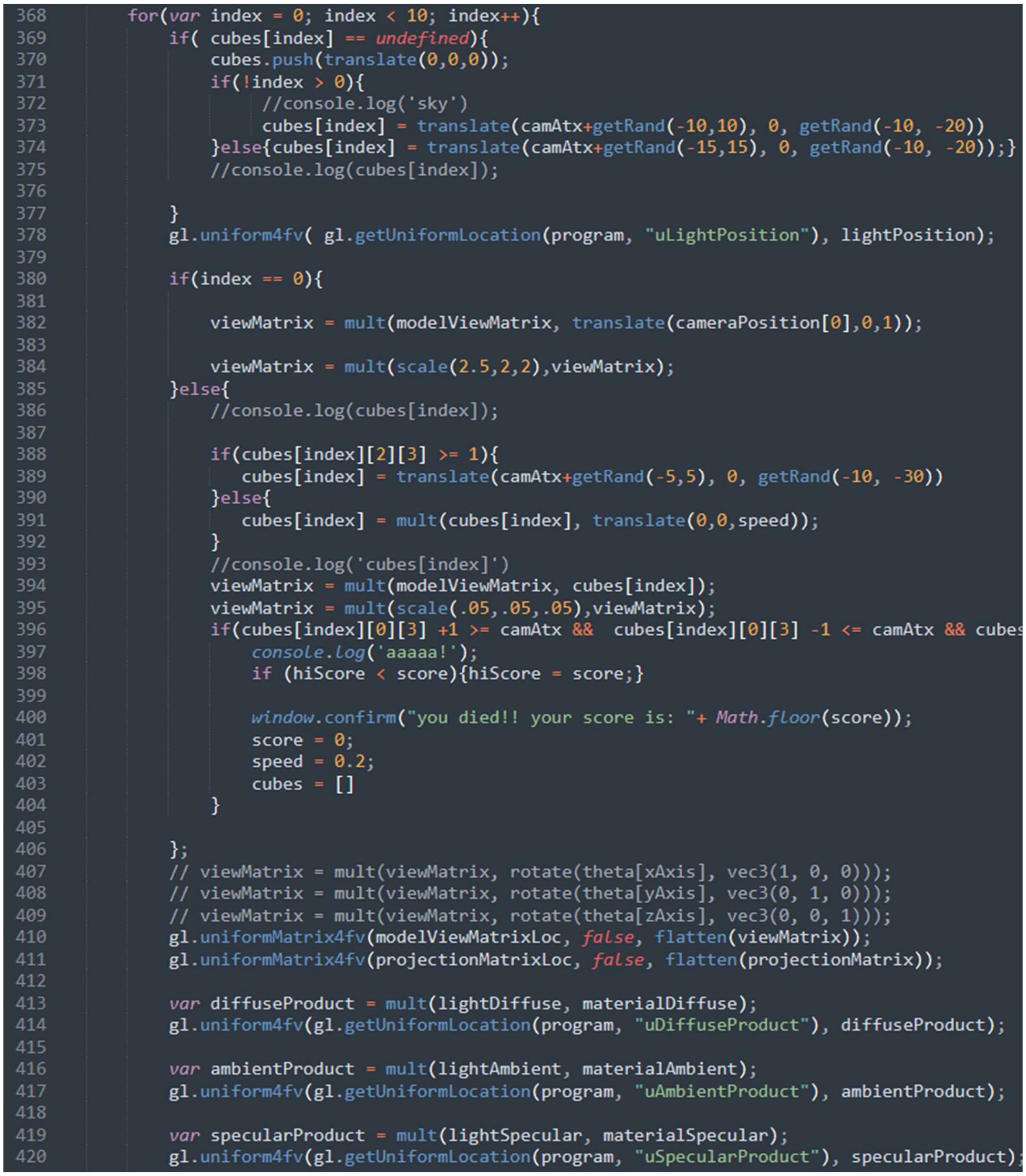
```
        var zFar = 30; //Change the value
        projectionMatrix = perspective(fieldOfView, aspect, zNear, zFar);

       //set event to the buttons
        window.addEventListener('keydown', this.checkKey);

        render();
    }

    function cubeBuffer(){
        gl.useProgram(program);
        var nBuffer = gl.createBuffer();
        gl.bindBuffer(gl.ARRAY_BUFFER, nBuffer);
        gl.bufferData(gl.ARRAY_BUFFER, flatten(normalsArray), gl.STATIC_DRAW);
        var normalLoc = gl.getAttribLocation(program, "aNormal");
        gl.vertexAttribPointer(normalLoc, 3, gl.FLOAT, false, 0, 0);
        gl.enableVertexAttribArray(normalLoc);

        //Create buffer for vertex
        var vBuffer = gl.createBuffer();
        gl.bindBuffer(gl.ARRAY_BUFFER, vBuffer);
        gl.bufferData(gl.ARRAY_BUFFER, flatten(positionsArray), gl.STATIC_DRAW);
        var positionLoc = gl.getAttribLocation(program, "aPosition");
        gl.vertexAttribPointer(positionLoc, 3, gl.FLOAT, false, 0, 0);
        gl.enableVertexAttribArray(positionLoc);

        //Create buffer for texture coordinate
        var tBuffer = gl.createBuffer();
        gl.bindBuffer(gl.ARRAY_BUFFER, tBuffer);
        gl.bufferData(gl.ARRAY_BUFFER, texCoordsArray, gl.STATIC_DRAW);
        var texCoordLoc = gl.getAttribLocation(program, "aTexCoord");
        gl.vertexAttribPointer(texCoordLoc, 2, gl.FLOAT, false, 0, 0);
        gl.enableVertexAttribArray(texCoordLoc);
        //load an image
        var image = document.getElementById("texImage");
        configureTexture(image);
        modelViewMatrixLoc = gl.getUniformLocation(program, "uModelViewMatrix");
        projectionMatrixLoc = gl.getUniformLocation(program, "uProjectionMatrix");
    }
    var positionsArrayP = [
        vec3(-0.25, -0.25,   0.26),
        vec3(0,      -0.1,      0.25),
        vec3(0,     -0.1,  -0.25),

        vec3(0.25, -0.25,   0.26),
        vec3(0,      -0.1,      0.25),
        vec3(0,     -0.1,  -0.25),

        vec3(0,     -0.1,  0.25),
        vec3(0.25, -0.25,  0.26),
        vec3(-0.25, -0.25,   0.26)
    ]
    var normalsArrayP = [
```



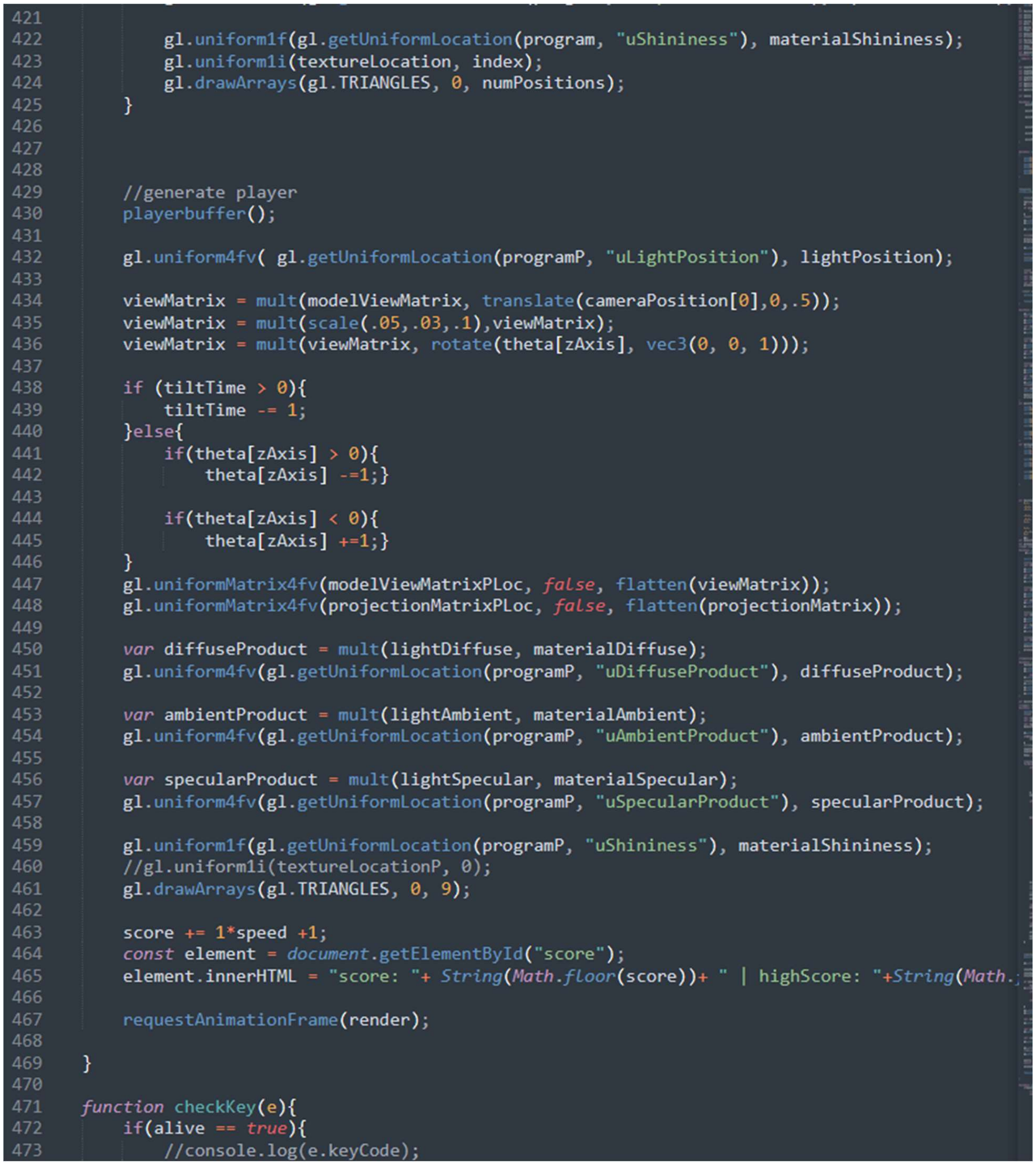
```
    vec3(-1, 1, 0),
    vec3(-1, 1, 0),
    vec3(-1, 1, 0),

    vec3(1, 1, 0),
    vec3(1, 1, 0),
    vec3(1, 1, 0),

    vec3(0, 0, 1),
    vec3(0, 0, 1),
    vec3(0, 0, 1)

]




var texCoordsArrayP = new Float32Array([
    0,1,
    .5,0,
    .5,1,

    .5,1,
    .5,0,
    1,1,

    1,1,
    .5,.5,
    0,1
    ]);
var modelViewMatrixPLoc
var projectionMatrixPLoc;
//load player
function playerbuffer(){
     gl.useProgram(programP);

    colorCube()
    //Create buffer for normals
    var nBufferP = gl.createBuffer();
    gl.bindBuffer(gl.ARRAY_BUFFER, nBufferP);
    gl.bufferData(gl.ARRAY_BUFFER, flatten(normalsArrayP), gl.STATIC_DRAW);
    var normalLocP = gl.getAttribLocation(programP, "aNormal");
    gl.vertexAttribPointer(normalLocP, 3, gl.FLOAT, false, 0, 0);
    gl.enableVertexAttribArray(normalLocP);

    //Create buffer for vertex
    var vBufferP = gl.createBuffer();
    gl.bindBuffer(gl.ARRAY_BUFFER, vBufferP);
    gl.bufferData(gl.ARRAY_BUFFER, flatten(positionsArrayP), gl.STATIC_DRAW);
    var positionLocP = gl.getAttribLocation(programP, "aPosition");
    gl.vertexAttribPointer(positionLocP, 3, gl.FLOAT, false, 0, 0);
    gl.enableVertexAttribArray(positionLocP);

```



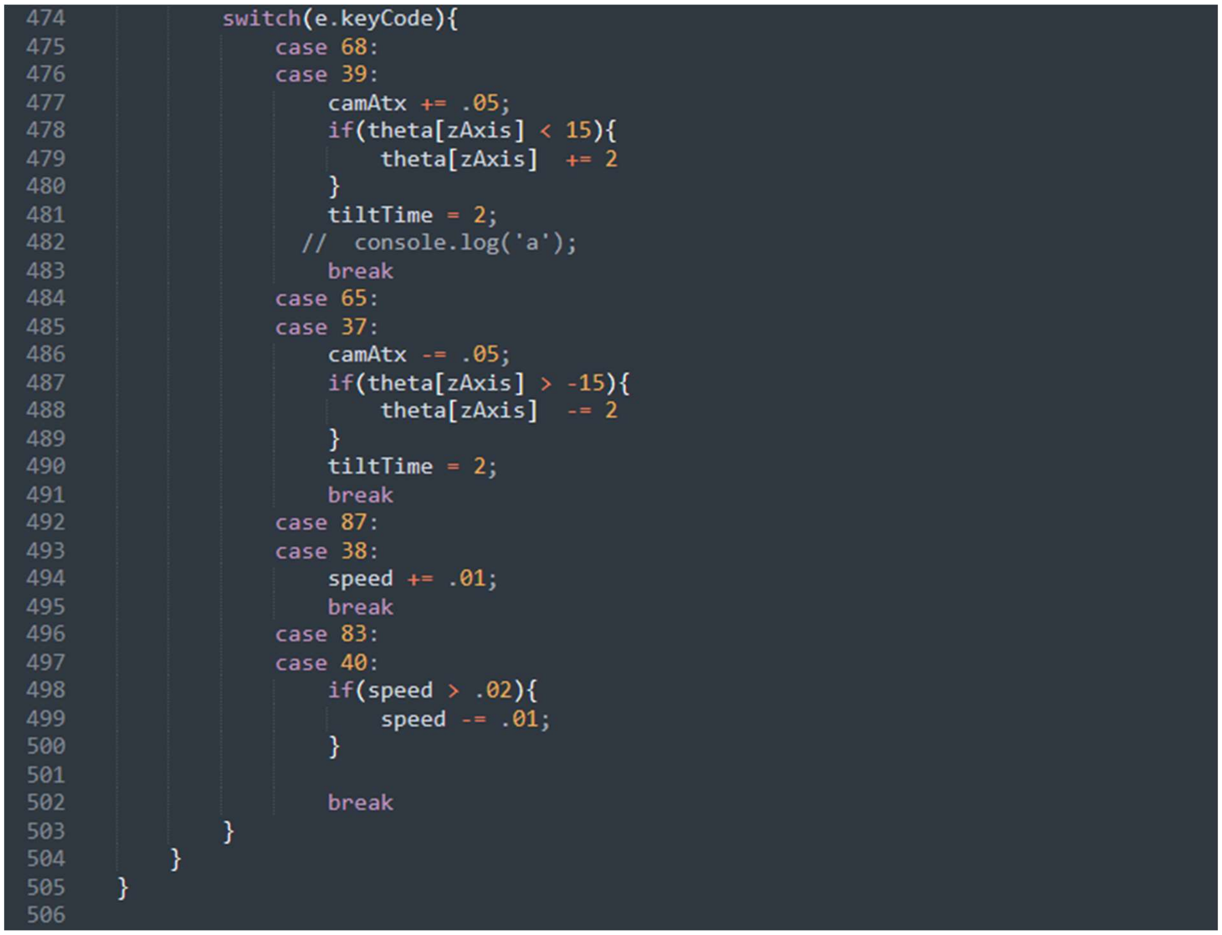
```
    //Create buffer for texture coordinate
    var tBufferP = gl.createBuffer();
    gl.bindBuffer(gl.ARRAY_BUFFER, tBufferP);
    gl.bufferData(gl.ARRAY_BUFFER, texCoordsArrayP, gl.STATIC_DRAW);
    var texCoordLocP = gl.getAttribLocation(programP, "aTexCoord");
    gl.vertexAttribPointer(texCoordLocP, 2, gl.FLOAT, false, 0, 0);
    gl.enableVertexAttribArray(texCoordLocP);

    //load an image
    var imageP = document.getElementById("playerImage");
    configureTexture(imageP);
    modelViewMatrixPLoc = gl.getUniformLocation(programP, "uModelViewMatrix");
    projectionMatrixPLoc = gl.getUniformLocation(programP, "uProjectionMatrix");
}

//function for setting the texture
function configureTexture(image){
    texture = gl.createTexture();
    gl.bindTexture(gl.TEXTURE_2D, texture);
    gl.texImage2D(gl.TEXTURE_2D, 0, gl.RGBA, 1, 1, 0, gl.RGBA, gl.UNSIGNED_BYTE,
        new Uint8Array([0, 0, 255, 255]));
    gl.texImage2D(gl.TEXTURE_2D, 0, gl.RGBA, gl.RGBA, gl.UNSIGNED_BYTE, image);
    //generate the Mipmap
    gl.generateMipmap(gl.TEXTURE_2D);
    gl.bindTexture(gl.TEXTURE_2D, texture);
    gl.texParameteri(gl.TEXTURE_2D, gl.TEXTURE_MIN_FILTER, gl.LINEAR_MIPMAP_NEAREST);
    gl.texParameteri(gl.TEXTURE_2D, gl.TEXTURE_MAG_FILTER, gl.LINEAR);
}

var tiltTime = 0;
//render function
function render(){
    cubeBuffer();
    //set the model-view matrix
    var cameraPosition = vec3(camAtx, 0, 1);
    var up = vec3(0.0, 1.0, 0.0);
    var target = vec3(camAtx, 0.0, 0.0);
    modelViewMatrix = lookAt(cameraPosition, target, up);

    gl.clear(gl.COLOR_BUFFER_BIT | gl.DEPTH_BUFFER_BIT);
    if(flag){
        theta[axis] += 0.3;  //Change the value
    }
    //rotating the light
    lightPosition[0] = 5.5 ;
    lightPosition[2] = 5.5;
    time += .5; //Change the value

    //generate two cubes, one is closer to the viewer and the other farther from the viewe
    for(var index = 0; index < 10; index++){
        if( cubes[index] == undefined){
            cubes.push(translate(0,0,0));
            if(!index > 0){
```



```
    for(var index = 0; index < 10; index++){
        if( cubes[index] == undefined){
            cubes.push(translate(0,0,0));
            if(!index > 0){
                 //console.log('sky')
                 cubes[index] = translate(camAtx+getRand(-10,10), 0, getRand(-10, -20))
            }else{cubes[index] = translate(camAtx+getRand(-15,15), 0, getRand(-10, -20));}
            //console.log(cubes[index]);

        }
        gl.uniform4fv( gl.getUniformLocation(program, "uLightPosition"), lightPosition);

        if(index == 0){

            viewMatrix = mult(modelViewMatrix, translate(cameraPosition[0],0,1));

            viewMatrix = mult(scale(2.5,2,2),viewMatrix);
        }else{
            //console.log(cubes[index]);

            if(cubes[index][2][3] >= 1){
               cubes[index] = translate(camAtx+getRand(-5,5), 0, getRand(-10, -30))
            }else{
               cubes[index] = mult(cubes[index], translate(0,0,speed));
            }
            //console.log('cubes[index]')
            viewMatrix = mult(modelViewMatrix, cubes[index]);
            viewMatrix = mult(scale(.05,.05,.05),viewMatrix);
            if(cubes[index][0][3] +1 >= camAtx &&  cubes[index][0][3] -1 <= camAtx && cubes
                console.log('aaaaa!');
                if (hiScore < score){hiScore = score;}

                window.confirm("you died!! your score is: "+ Math.floor(score));
                score = 0;
                speed = 0.2;
                cubes = []
            }

        };
        // viewMatrix = mult(viewMatrix, rotate(theta[xAxis], vec3(1, 0, 0)));
        // viewMatrix = mult(viewMatrix, rotate(theta[yAxis], vec3(0, 1, 0)));
        // viewMatrix = mult(viewMatrix, rotate(theta[zAxis], vec3(0, 0, 1)));
        gl.uniformMatrix4fv(modelViewMatrixLoc, false, flatten(viewMatrix));
        gl.uniformMatrix4fv(projectionMatrixLoc, false, flatten(projectionMatrix));

        var diffuseProduct = mult(lightDiffuse, materialDiffuse);
        gl.uniform4fv(gl.getUniformLocation(program, "uDiffuseProduct"), diffuseProduct);

        var ambientProduct = mult(lightAmbient, materialAmbient);
        gl.uniform4fv(gl.getUniformLocation(program, "uAmbientProduct"), ambientProduct);

        var specularProduct = mult(lightSpecular, materialSpecular);
        gl.uniform4fv(gl.getUniformLocation(program, "uSpecularProduct"), specularProduct);
```



```
        gl.uniform1f(gl.getUniformLocation(program, "uShininess"), materialShininess);
        gl.uniform1i(textureLocation, index);
        gl.drawArrays(gl.TRIANGLES, 0, numPositions);
    }



    //generate player
    playerbuffer();

    gl.uniform4fv( gl.getUniformLocation(programP, "uLightPosition"), lightPosition);

    viewMatrix = mult(modelViewMatrix, translate(cameraPosition[0],0,.5));
    viewMatrix = mult(scale(.05,.03,.1),viewMatrix);
    viewMatrix = mult(viewMatrix, rotate(theta[zAxis], vec3(0, 0, 1)));

    if (tiltTime > 0){
        tiltTime -= 1;
    }else{
        if(theta[zAxis] > 0){
            theta[zAxis] -=1;}

        if(theta[zAxis] < 0){
            theta[zAxis] +=1;}
    }
    gl.uniformMatrix4fv(modelViewMatrixPLoc, false, flatten(viewMatrix));
    gl.uniformMatrix4fv(projectionMatrixPLoc, false, flatten(projectionMatrix));

    var diffuseProduct = mult(lightDiffuse, materialDiffuse);
    gl.uniform4fv(gl.getUniformLocation(programP, "uDiffuseProduct"), diffuseProduct);

    var ambientProduct = mult(lightAmbient, materialAmbient);
    gl.uniform4fv(gl.getUniformLocation(programP, "uAmbientProduct"), ambientProduct);

    var specularProduct = mult(lightSpecular, materialSpecular);
    gl.uniform4fv(gl.getUniformLocation(programP, "uSpecularProduct"), specularProduct);

    gl.uniform1f(gl.getUniformLocation(programP, "uShininess"), materialShininess);
    //gl.uniform1i(textureLocationP, 0);
    gl.drawArrays(gl.TRIANGLES, 0, 9);

    score += 1*speed +1;
    const element = document.getElementById("score");
    element.innerHTML = "score: "+ String(Math.floor(score))+ " | highScore: "+String(Math.

    requestAnimationFrame(render);

}

function checkKey(e){
    if(alive == true){
        //console.log(e.keyCode);
```



```
        switch(e.keyCode){
            case 68:
            case 39:
                camAtx += .05;
                if(theta[zAxis] < 15){
                    theta[zAxis]  += 2
                }
                tiltTime = 2;
              //  console.log('a');
                break
            case 65:
            case 37:
                camAtx -= .05;
                if(theta[zAxis] > -15){
                    theta[zAxis]  -= 2
                }
                tiltTime = 2;
                break
            case 87:
            case 38:
                speed += .01;
                break
            case 83:
            case 40:
                if(speed > .02){
                    speed -= .01;
                }

                break
        }
    }
}
```