GAME 3D WEBGL

1. HTML

```
  <html>
    <head>
        <script type="text/javascript"
src="libs/TweenMax.min.js"></script>
        <script type="text/javascript"
src="libs/three.min.js"></script>
        <script type="text/javascript" src="game.js"></script>
        <title>Plane Run </title>
        <!-- <link
href='https://fonts.googleapis.com/css?family=Playfair+Display:400,700
,700italic' rel='stylesheet' type='text/css'> -->
        <link rel="stylesheet" type="text/css" href="game.css" />
    </head>
    <body>
        <!-- <iframe src = "sounds/Strategy.mp3"  type="audio/mp3"
allow="autoplay" id="audio" style="display: none"> </iframe> -->
        <audio autoplay loop id = "backAudio"
src="sounds/background.mp3"  type="audio/mp3"></audio>
        <div class="mainClass" id="mainClass">

            <!-- This part will contain the level and score -->
            <div class = "rightSideParamters"><!-- This part will
contain the stage and points -->
                <div class="parameterDisplay" id="level">
                    <div class="parameterHeading">Stage</div>
                    <div class="displayStageAndBullets"
id="currStage">1</div>
                    <svg class="stageBulletContainer"
id="stageContainer" viewbox="0 0 200 200">
                        <circle id="stageContainerBgr" r="80" cx="100"
cy="100" fill="none" stroke="#e6ac55" stroke-width="24px" />
                        <circle id="stageContainerStroke" r="80"
cx="100" cy="100" fill="none" stroke="#2b0261" stroke-width="14px"
stroke-dasharray="502" />
                    </svg>
                </div>
                <div class="parameterDisplay" id="dist">
                    <div class="parameterHeading">Points</div>
                    <div class="displayPoints"
id="distValue">000</div>
                </div>
            </div>

            <!-- This part will contain the health bar and number of
bullets -->
            <div class = "leftSideParameters"><!-- This part will
contain the health meter and number of bullets -->
                <div class="parameterDisplay" id="health">
                    <div class="parameterHeading">Health</div>
                    <div class="healthMeter" id="healthMeter">
```

```
                        <div class="healthMeterFill"
id="healthMeterFill"></div>
                    </div>
                </div>
                <div class="parameterDisplay" id="bullets">
                    <div class="parameterHeading">bullets</div>
                    <div class="displayStageAndBullets"
id="bulletCount">2</div>
                    <svg class="stageBulletContainer"
id="bulletContainer" viewbox="0 0 200 200">
                        <circle id="bulletContainerBgr" r="80"
cx="100" cy="100" fill="none" stroke="#7d4d05" stroke-width="24px" />
                    </svg>
                </div>
            </div>

            <div class="gameNameHeading">  <!-- This part will contain
game name -->
                <top>The    </top>Plane Run
                <bottom>How far can you go ??</bottom>
            </div>

            <div class="mainScene" id="mainScene"></div>
            <div class="clickToReplay" id="clickToReplayMsg">Click or
Press any key to Replay</div>
            <div class="clickToStart" id="clickToStartMsg">Press any
key to Start!!</div>
            <div class="howToPlay" id="howToPlayMsg">Collect the green
coloured jewels to gain health!!<br><next>Stay away from the
Meteors!!<br><next></next></next>Press 'X' to shoot
bullets!!</next></next></div>
        </div>
    </body>
</html>
```

2. Css

```
/*This containter contains the game's elements - plane, sky,
clouds

id value for this is = "mainScene"

this can be accessed in game.js using
document.getElementById()*/

.mainScene {

     position: absolute;

     overflow: hidden;

     width: 100%;

     height: 100%;

}


/*This container contains the style sheet for the name of the
game*/

.gameNameHeading {

     position: absolute;

     top: 10;

     left: 0;

     width: 100%;

     text-align: center;

     pointer-events: none;

     color: #3e038b;

     font-family: 'Gill Sans', 'Gill Sans MT', Calibri,
'Trebuchet MS', sans-serif;

     font-style: normal;

     font-size: 3em;

     letter-spacing: -0.03em;

}


/*Style sheet for writing 'the'*/

.gameNameHeading top{

     font-style: italic;
```

```
      color: #3a23a0;
      display: block;
      letter-spacing: 0px;
      font-family: 'Segoe UI', Tahoma, Geneva, Verdana, sans-
serif;
      font-size: 0.4em;
      left: 0;
      top: 0;
      bottom: -2;
}

/*Style sheet for writing 'How far can you go??'*/
.gameNameHeading bottom{
      font-style: italic;
      color: #880267;
      display: block;
      letter-spacing: 0px;
      font-family: Verdana, Geneva, Tahoma, sans-serif;
      font-size: 0.3em;
      left: 0;
      top: 0;
}

/*This container contains the style sheet for current level and
score*/
.rightSideParamters {
      position: absolute;
      right: 50;
      top: 6vh;
      text-align: center;
      pointer-events: none;
}
```

```
/*This container contains the style sheet for current number of
bullets and Energy left.*/
.leftSideParameters {
	position: absolute;
	left: 50;
	top: 6vh;
	text-align: center;
	pointer-events: none;
}

/*This stye sheet is for styling all the following elements -
points, heath bar, stage, bullets*/
.parameterDisplay {
	position: relative;
	display: inline-block;
	padding: 0 1em;
	vertical-align: top;
}

/*This style sheet is for creating the separator bars between
the various score/health/bullet related elements*/
.parameterDisplay:nth-child(2) {
	border-right: 2px solid #881c37;
	border-left: 2px solid #881c37;
}

/*This style sheet is for designing the text that is used in
each score/health/bullet related element's heading*/
.parameterHeading {
	font-size: 16px;
	font-weight: bold;
```

```
	position: relative;
	margin: 0 0 0.5em 0;
	text-align: center;
	letter-spacing: 4px;
	text-transform: uppercase;
	color: #572700;
}

/*This style sheet is for displaying the numerical text for
current Stage and no. of bullets left*/
.displayStageAndBullets {
	font-size: 26px;
	font-family: 'Comic Sans MS';
	font-weight: bold;
	color: #4d0049;
}

/*This style sheet is for displaying the numerical text for
current no. of points the player has scored.*/
.displayPoints {
	font-size: 30px;
	font-family: 'Comic Sans MS';
	font-weight: bold;
	color: #680139;
}

/*This style sheet is for displaying the health meter -
it shows the remaining amount of health the player has*/
.healthMeter {
	position: relative;
	border-radius: 3px;
```

```
      background-color: #eeebe7;
      margin-top: 20px;
      color: #4d0049;
      width: 60px;
      height: 8px;
}

/*This style sheet iis for filling the health meter with
the appropriate proportion of health left for the player*/
.healthMeterFill {
      position: absolute;
      top: 0;
      right: 0;
      bottom: 0;
      left: 0;
      margin: 2px;
      background-color: #f25346;
      -webkit-animation-name: none;
      animation-name: none;
      -webkit-animation-duration: 150ms;
      animation-duration: 150ms;
      -webkit-animation-iteration-count: infinite;
      animation-iteration-count: infinite;
}

/*This style sheet is for drawing the circular border around
the stage number and number of bullets left*/
.stageBulletContainer {
      position: absolute;
      left: 50%;
      width: 46px;
```

```
      margin: -37px 0 0 -23px;
      -webkit-transform: rotate(-90deg);
      transform: rotate(-90deg);
}

/*Style sheet for the replay message that is displayed
onto the screen when the player loses*/
.clickToReplay {
      position: absolute;
      bottom: 52vh;
      display: none;
      text-indent: 0.5em;
      letter-spacing: 0.5em;
      color: #0a0097;
      font-weight: bold;
      left: 0;
      width: 100%;
      font-size: 1.5vw;
      text-align: center;
      text-transform: uppercase;
      pointer-events: none;
}

/*Style sheet for the click to start message when the
player opens the game for the first time*/
.clickToStart {
      position: absolute;
      bottom: 52vh;
      display: none;
      text-indent: 0.5em;
      letter-spacing: 0.5em;
```

```
      color: #0a0097;
      font-weight: bold;
      left: 0;
      width: 100%;
      font-size: 2.5vw;
      text-align: center;
      pointer-events: none;
}

/*This contains the style sheet for displaying how to play
instructions on the screen*/
.howToPlay {
      position: absolute;
      margin-top: 87vh ;
      margin-left: 55vh;
      text-align: center;
      font-weight: bold;
      font-style: oblique;
      font-size: 1.2em;
      font-family: 'Lucida Sans', 'Lucida Sans Regular', 'Lucida
Grande', 'Lucida Sans Unicode', Geneva, Verdana, sans-serif;
      color: #0b46b3;
      letter-spacing: 0.18em;
      pointer-events: none;
}

/*Style sheet for displaying how to play instructions
on the screen*/
.howToPlay next{
      color: #ff0a0a;
      white-space: nowrap;
      text-align: center;
```

```
}


/*Style sheet for displaying how to play instructions
on the screen- Press X to fire*/
.howToPlay next next {
	color: #019c35;
	text-align: center;
}


/*Outermost class containing all the elements*/
.mainClass {
	position: absolute;
	width: 100%;
	height: 100%;
	background: -webkit-linear-gradient(rgb(14, 112, 96),
rgb(3, 62, 99));
	background: linear-gradient(rgb(134, 180, 159), rgb(192,
115, 43));
}


/*This keyframe animation effect is used for creating
a flickering effect when health is low on health bar*/
@keyframes flickering {
	0% { opacity: 1; }
	50% { opacity: 0; }
	100% { opacity: 1; }
}
```

3. Js

```
/*
**********************************************************
**********

 * ***************LIST OF
VARIABLES**********************************

 *
**********************************************************
*********

  */


/* This variable is the main controller for the game. It
contains various attributes that are

relevant to the game. All of them are initialized in the
generateNewGame() function */

var controller;


/* Basic three js variable that defines the scene/setting
(initialized using THREE.Scene)  */

var scene;


/* Basic three js variable that defines the perpectivCamera
(initalized using THREE.PerspectiveCamera) */

var perspCam;


/* Basic three js variable that defines the webGlRenderer
(initalized using THREE.WebGLRenderer) */

var webGlRenderer;


/* Basic three js variable that will contain the renderer's
domElement

It is the link between our HTML element (mainScene) and
three js */

var container;
```

```
/* Variables used while generating the lights for the game
setting  */

var ambLight, hemiLight, lightOfShadow;


/* Time value of current frame */

var currFrameTime = new Date().getTime();


/* Time value of previous frame */

var prevFrameTime = new Date().getTime();


/* This variable is a time element (time elapsed between
consecutive animation frames).

It is computed at the start of each frame using the
Date().getTime() function */

var dT = 0;


/* This array contains all the active meteors in the game
setting */

var meteorsPool = [];


/* This array contains all the active fragments(generated
when a meteor

  explodes when either the aircraft or a bullet hits it) in
the game setting */

var fragmentsPool = [];


/* This array contains all the active bullets in the game
setting

Bullets that either hit a meteor or move out of the window
are

deleted from this array immediately   */
```

```
var bulletsInUse = [];


/* This is a switch variable (bool - 0 or 1) */

var canFireBullet = 1;


/* Variable that defines the angle (in degrees) span that
the camera is able to capture

in our current scene. Needed during camera initialization
*/

var viewSpan = 55;


/* Following variables represent the screen height and
width  */

var HEIGHT = window.innerHeight;

var WIDTH = window.innerWidth;


/* Defines the aspect ratio of our screen. Needed during
camera initialization  */

var aspRatio = WIDTH/HEIGHT;


/* This denotes the mouse pointer position. The aircraft is
moved based on

changes in currMousePtrLoc.x and currMousePtrLoc.y  */

var currMousePtrLoc = { x: 0, y: 0 };


/* List of colors that we will be using throughout..

this variable is only for easier coding which otherwise
holds no significance */

var usefulColours = { mediumAqua: 0x66CDAA,
lightOceanGreen: 0x20B2AA, limeGreen: 0x1CE678,
greenYellow: 0xADFF2F, lightBlue: 0x1E90FF, darkBlue:
0x00008B, red:0xf25346, white:0xd8d0d1, brown:0x59332e,
```

```
blue:0x68c3c0, silver:0x808080,  black: 0x000000, violet:
0x8A2BE2};


/* Switch (bool - 0 or 1) variable used to determine when
to display

the 'click to start' message  */

var gameStarted = 0;


/*Variables for various objects in the game environment.
Their names are self explanatory*/

var ocean, aircraft, sky;


/*The following are variables that are associated with
elements from the index.html file like

  current points, no. of bullets etc. These are used as
handles so that the internal changes in

  the game parameters get reflected onto the UI */

var currPoints, healthMeterFill, replayMessage, stageField,
stageContainer, bulletVal, startMessage;


/* *

 *
**********************************************************
*****

 * ************LIST OF FUNCTIONS FOR
INITIALIZING*****************

 * ****************THE THREEJS
ENVIRONMENT************************

  */


/* This function initializes the threejs scene, camera and
renderer

so that we can render the scene with the camera  */
```

```
function generateScene() {
  //create scene
  scene = new THREE.Scene();
  //create camera
  perspCam = new THREE.PerspectiveCamera(viewSpan,
aspRatio, 0.05, 12000);
  //add fog to the scene for realistic effect
  scene.fog = new THREE.Fog(0xAAF7D9, 99,1000);
  //set camera position
  perspCam.position.x = 0;
  perspCam.position.z = 200;
  perspCam.position.y = controller.aircraftDefaultHeight;
  //create renderer
  webGlRenderer = new THREE.WebGLRenderer({ stencil: true,
alpha: true, logarithmicDepthBuffer: false, antialias: true
});
  webGlRenderer.setSize(WIDTH, HEIGHT);
  //specify that renderer has to render shadow maps
  webGlRenderer.shadowMap.enabled = true;
  //initialize container
  container = document.getElementById('mainScene');
  container.appendChild(webGlRenderer.domElement);
}


/* This function lights the scene using three js library
functions and populates
the light object's required attributes */
function generateLighting() {
  ambLight = new THREE.AmbientLight(0xdc8874, .5);
  hemiLight = new THREE.HemisphereLight(0xaaaaaa,0x000000,
.9);
```

```
  lightOfShadow = new THREE.DirectionalLight(0xffffff, .9);
  lightOfShadow.position.set(150, 350, 350);
  lightOfShadow.castShadow = true;
  lightOfShadow.shadow.camera.right = 400;
  lightOfShadow.shadow.camera.top = 400;
  lightOfShadow.shadow.camera.left = -400;
  lightOfShadow.shadow.camera.bottom = -400;
  lightOfShadow.shadow.camera.near = 1;
  lightOfShadow.shadow.camera.far = 1000;
  //below 2 attributes are needed to specify ratio of the
shadow length
  //w.r.t the length of in-game elements
  lightOfShadow.shadow.mapSize.width = 4096;
  lightOfShadow.shadow.mapSize.height = 4096;


  scene.add(hemiLight);
  scene.add(lightOfShadow);
  scene.add(ambLight);


}


/* This function is triggered/executed whenever the game is
restarted.
It initializes the attributes of the controller variable
and resets
the stage and bullets count on the UI  */
function generateNewGame(){
  controller = {
        stateOfGame : "notStarted",


        /* Variables related to speed of game and plane */
```

```
        baseSpeed:.00035,
        startSpeed:.00035,
        distToUpdateSpeed:100,
        lastUpdatedSpeed:0,
        speed:0,
        targetBaseSpeed:.00035,
        increaseSpeedWithLevel:.000005,
        increaseSpeedWithTime:.0000025,

        /*  Variables that are used for storing the display
variables of the HTML file. */
        pointsScored:0,
        ratioOfSpeedEnergy:3,
        ratioOfSpeedDistance:50,
        stage:1,
        health:100,

        /* Variables used for updating the stage of the
game. */
        stageLastUpdate:0,
        distanceForStageUpdate:1000,

        /* Variable to store properties of the aircraft */
        aircraftDefaultHeight:100,
        aircraftAmplitudeHt:80,
        aircraftAmplitudeWdth:75,
        //Following sensitivity variables were created to
adjust the result of mathematical calculations for
        //rotation and displacement of aircraft mesh.
        aircraftMovementSensi:0.005,
        aircraftRotationXSensi:0.0008,
```

```
        aircraftRotationZSensi:0.0004,
        aircraftFallSpeed:.001,
        minAircraftSpeed:1.2,
        maxAircraftSpeed:1.6,
        aircraftSpeed:0,

        /* Variables related to collision of aircraft. */
        aircraftCollisionXDisplacement:0,
        aircraftCollisionXSpeed:0,
        aircraftCollisionYSpeed:0,
        aircraftCollisionYSpeed:0,

        /* Variables related to bullets. */
        bulletCount:2,
        bulletSpeed:100,

        /* Variables related to Ocean */
        radiusOfSea:600,
        lengthOfSea:800,
        minSpeedOfWaves : 0.001,
        maxSpeedOfWaves : 0.003,
        minSizeOfWaves : 5,
        maxSizeOfWaves : 20,


        /*  Variables made for camera positioning and
sensitivity.  */
        farPosOfCamera:500,
        nearPosOfCamera:150,
        cameraSensivity:0.002,
```

```
        jewelDistanceToler:15,
        speedOfJewel:.5,
        jewelSpawnDistance:100,
        valueOfJewel:3,
        lastSpawnOfJewel:0,

        /* Variables related to enemy meteors. */
        healthLossByMeteor:10,
        meteorsSpeed:.6,
        distanceToleranceInMeteor:10,
        lastSpawnOfMeteor:0,
        meteorSpawnDistance:50,
        };
  stageField.innerHTML = Math.floor(controller.stage);
  bulletVal.innerHTML = Math.floor(controller.bulletCount);
}



/* *
 * ******************************************
 * ******FUNCTIONS FOR CREATING THE VARIOUS*****
 * ******ELEMENTS IN THE SCENE LIKE BULLETS,****
 * ******AIRCRAFT,SKY,JEWELS,OCEAN, CLOUD,******
 * ******METEOR,FRAGMENTS-----------------******
  */
```

```
/*************** 1: BULLET ************ */

/* ******************Function related to bullet
creation.******************* */

/* This function is used to create mesh of bullet object
(bullet constructor) */

Bullet = function(){

    this.mesh = new THREE.Mesh(new THREE.SphereGeometry(3,
8, 8), new THREE.MeshBasicMaterial( {color:
usefulColours.black} ));

}


/* This function creates an individual bullet. */

function generateBullet(){

  if(controller.bulletCount > 0)

  {

    var fireSound = new Audio('sounds/bulletFire.wav');

    fireSound.play();

    fireSound.volume = 0.6;

    controller.bulletCount--;

    if(controller.bulletCount == 0)

      bulletVal.style.animationName = 'flickering';

    bulletVal.innerHTML =
Math.floor(controller.bulletCount);

    var newBullet = new Bullet();


newBullet.mesh.position.copy(aircraft.rotor.getWorldPositio
n());

    //console.log("\nAircraft position : " +
aircraft.rotor.getWorldPosition().x);

    //console.log("\nnew BUllet position : " +
newBullet.mesh.position.x);

    bulletsInUse.push(newBullet);

    scene.add(newBullet.mesh);
```

```
  }
}


/* This function is used to update the postion of the
bullet. It also checks for collision of bullets with enemy
Meteors. */
function updateBullet(){


  for(var i=0; i<bulletsInUse.length;i++){
    //console.log(bulletsInUse[i]);
    //console.log(bulletsInUse[i].mesh.position.x + " " +
i);
    var bullet = bulletsInUse[i];
    bulletsInUse[i].mesh.position.x += 8;


    for (var j=0; j<meteorsHolder.meteorsInUse.length;
j++){
        var meteor = meteorsHolder.meteorsInUse[j];
        var position_diff =
bullet.mesh.position.clone().sub(meteor.mesh.position.clone
());
        var diff = position_diff.length();
        if (diff<controller.distanceToleranceInMeteor){

fragmentsHolder.spawnFragments(meteor.mesh.position.clone()
, 15, usefulColours.violet, 3);

meteorsPool.unshift(meteorsHolder.meteorsInUse.splice(j,1)[
0]);
          meteorsHolder.mesh.remove(meteor.mesh);
          scene.remove(bulletsInUse[i].mesh);
          bulletsInUse.splice(i,1);
          controller.pointsScored += 100;
```

```
          var meteor_bulletSound = new
Audio('sounds/rockBreaking.wav');
          meteor_bulletSound.play();
          meteor_bulletSound.volume = 1;
        }
        else{

        }
    }
  }
}

/*************** 2: AIRCRAFT *********** */
/* ******************Function related to aircraft
creation.******************** */
/* This function is used to create mesh of bullet object.
(aircraft constructor)*/
var Aircraft = function(){
  //1.Define the mesh as a generic 3D Object
    this.mesh = new THREE.Object3D();
    this.mesh.castShadow = true;
    this.mesh.receiveShadow = true;
    this.mesh.name = "aircraft";

  //2.Create the fuselage
    //define geometry
    var fuselageGeometry = new THREE.BoxGeometry(80,50,50);
//generates cuboid with l,b,h=80,50,50
    //set its vertices
    fuselageGeometry.vertices[4].y-=10;
    fuselageGeometry.vertices[5].y-=10;
```

```
    fuselageGeometry.vertices[6].y+=30;

    fuselageGeometry.vertices[7].y+=30;

    fuselageGeometry.vertices[4].z+=20;

    fuselageGeometry.vertices[5].z-=20;

    fuselageGeometry.vertices[6].z+=20;

    fuselageGeometry.vertices[7].z-=20;

    //define material

    var fuselageMaterial = new THREE.MeshPhongMaterial({
emissiveIntensity: 0.9, color:usefulColours.darkBlue });

    //create the mesh using geometry,material

    var fuselage = new THREE.Mesh(fuselageGeometry,
fuselageMaterial);

    fuselage.receiveShadow = true;

    fuselage.castShadow = true;

    this.mesh.add(fuselage);


  //3. Create the Nose of the Aircraft (the part before the
propelller)

    var noseGeometry = new THREE.BoxGeometry(20,50,50);

    var noseMaterial = new THREE.MeshPhongMaterial({
emissiveIntensity: 0.9, color:usefulColours.lightBlue });

    var nose = new THREE.Mesh(noseGeometry, noseMaterial);

    //set the x position for the nose relative to the
fuselage

    nose.position.x = 50;

    nose.castShadow = true;

    nose.receiveShadow = true;

    this.mesh.add(nose);


  //4. Create the tail of the Aircraft (Part where the
airline logo is usually located)

    var tailFinGeometry = new THREE.BoxGeometry(15,20,5);
```

```
    var tailFinMaterial = new THREE.MeshPhongMaterial({
emissiveIntensity: 0.9, color:usefulColours.lightBlue });

    var tailFin = new THREE.Mesh(tailFinGeometry,
tailFinMaterial);

    tailFin.receiveShadow = true;

    //set the tailFin position relative to the fuselage--
bit to the left and bit upward

    tailFin.position.set(-40,20,0); //set position(x,y,z)

    tailFin.castShadow = true;

    this.mesh.add(tailFin);


  //5. Create the side wing of the Aircraft

    //this single object will take care of both left wing
and right wing

    var wingGeometry = new THREE.BoxGeometry(30,5,120);

    var wingMaterial = new THREE.MeshPhongMaterial({
emissiveIntensity: 0.9, color:usefulColours.mediumAqua,
shading:THREE.FlatShading});

    var wing = new THREE.Mesh(wingGeometry, wingMaterial);

    wing.castShadow = true;

    wing.receiveShadow = true;

    //position is almost same relative to fuselage, but
just shifted a bit upward

    //the wing clearly is 1 single object and cuts through
the fuselage

    wing.position.set(0,15,0);

    this.mesh.add(wing);


  //6.1. Create the rotor/propelller of the Aircraft

    var rotorGeometry = new THREE.BoxGeometry(20,10,10);

    //rotor coordinates

    rotorGeometry.vertices[4].y-=5;
```

```
    rotorGeometry.vertices[5].y-=5;

    rotorGeometry.vertices[6].y+=5;

    rotorGeometry.vertices[7].y+=5;

    rotorGeometry.vertices[4].z+=5;

    rotorGeometry.vertices[5].z-=5;

    rotorGeometry.vertices[6].z+=5;

    rotorGeometry.vertices[7].z-=5;

    var rotorMaterial = new THREE.MeshPhongMaterial({
emissiveIntensity: 0.9, color:usefulColours.brown,
shading:THREE.FlatShading});

    this.rotor = new THREE.Mesh(rotorGeometry,
rotorMaterial);

    this.rotor.castShadow = true;

    this.rotor.receiveShadow = true;


    //6.2. Create the blades for the rotor

      //create a cuboid with long y value (long breadth)
and short x and z values

      var propBladeGeometry = new
THREE.BoxGeometry(1,80,10);

      var propBladeMaterial = new THREE.MeshPhongMaterial({
emissiveIntensity: 0.9, color:usefulColours.black,
shading:THREE.FlatShading});

      //first blade - vertical

      var vertPropBlade = new THREE.Mesh(propBladeGeometry,
propBladeMaterial);

      vertPropBlade.castShadow = true;

      vertPropBlade.receiveShadow = true;

      //blade position relative to rotor - very slightly
shifted to the right

      vertPropBlade.position.set(8,0,0);

      //second blade is basically the same (copy) of the
first blade
```

```
      var horizPropBlade = vertPropBlade.clone();
      //just need to rotate it by 90 degrees
      //NOTE: rotation about x axis (rotation on y-z plane)
      horizPropBlade.rotation.x = Math.PI/2;
      horizPropBlade.castShadow = true;
      horizPropBlade.receiveShadow = true;
      //add the blades to the rotor
      this.rotor.add(vertPropBlade);
      this.rotor.add(horizPropBlade);
    //rotor position relative to the fuselage
    this.rotor.position.set(60,0,0);
    //add the rotor to the aircraft's mesh
    this.mesh.add(this.rotor);


  //7. Design the wheels/landing gear for the aircraft
    //7.1 Add the right landing gear
      var landingGearGeometry = new
THREE.BoxGeometry(24,24,4);
      var landingGearMaterial = new
THREE.MeshPhongMaterial({ emissiveIntensity: 0.9,
color:usefulColours.black, shading:THREE.FlatShading});
      var rightMainlandingGear = new
THREE.Mesh(landingGearGeometry,landingGearMaterial);
      rightMainlandingGear.position.set(25,-28,25);
      //7.1.1 Add the axis for the landing gear
        var LandingGearAxisGeometry = new
THREE.BoxGeometry(10,10,6);
        var LandingGearAxisMaterial = new
THREE.MeshPhongMaterial({ emissiveIntensity: 0.9,
color:usefulColours.brown, shading:THREE.FlatShading});
        var LandingGearAxis = new
THREE.Mesh(LandingGearAxisGeometry,LandingGearAxisMaterial)
;
```

```
        rightMainlandingGear.add(LandingGearAxis);

      this.mesh.add(rightMainlandingGear);

    //7.2 Clone right tire to make left landin gear

      var leftMainlandingGear =
rightMainlandingGear.clone();

      leftMainlandingGear.position.z = -
rightMainlandingGear.position.z;

      this.mesh.add(leftMainlandingGear);

    //7.3 Clone the right tire to make the rear landing
gear

      var rearlandingGear = rightMainlandingGear.clone();

      //back gear is smaller than the front left and right

      //scaling by a factor of .5 along x,y,z directions

      rearlandingGear.scale.set(.5,.5,.5);

      rearlandingGear.position.set(-35,-5,0);

      this.mesh.add(rearlandingGear);

    //7.4 Add the chock/cover for the right gear

      var LandingGearChockGeometry = new
THREE.BoxGeometry(30,15,10);

      var LandingGearChockMaterial = new
THREE.MeshPhongMaterial({ emissiveIntensity: 0.9,
color:usefulColours.red, shading:THREE.FlatShading});

      var LandingGearRightChock = new
THREE.Mesh(LandingGearChockGeometry,LandingGearChockMateria
l);

      LandingGearRightChock.position.set(25,-20,25);

      this.mesh.add(LandingGearRightChock);

    //7.5 Clone the rear wheel chock to make left chock

      var LandingGearLeftChock =
LandingGearRightChock.clone();

      LandingGearLeftChock.position.z = -
LandingGearRightChock.position.z ;

      this.mesh.add(LandingGearLeftChock);
```

```
};


/* This function creates our aircraft. */
function generateAircraft(){
  aircraft = new Aircraft();
  //adjust the scale.. original aircraft became too big
compared to the game environment
  aircraft.mesh.scale.set(.25,.25,.25);
  //set the starting y corrdinate for the aircraft as the
controller's default height
  //this is used as a reference in updateAircraft()
  aircraft.mesh.position.y =
controller.aircraftDefaultHeight;
  scene.add(aircraft.mesh);
}


/* This function is used to update the postion of the
aircraft
  Also, it handles the collision of aircraft with meteors
  and appropriately triggers the code for generating meteor
  fragments upon collision */
function updateAircraft(){
  //modify aircraft speed (relative to the screen) when the
user moves the mouse in x direction
  //the game window zooms in/out when mouse pointer is
moved in x direction. to compensate for this
  //we need to change aircraft speed
  controller.aircraftSpeed =
transformValue(currMousePtrLoc.x,-
.5,.5,controller.minAircraftSpeed,
controller.maxAircraftSpeed);
```

```
  //calculate the aircraft displacement along the x
direction when mouse is moved horizontally
  var targetX = transformValue(currMousePtrLoc.x,-1,1,-
controller.aircraftAmplitudeWdth*0.7, -
controller.aircraftAmplitudeWdth);
  controller.aircraftCollisionXDisplacement +=
controller.aircraftCollisionXSpeed;
  targetX += controller.aircraftCollisionXDisplacement;


  //calculate the aircraft displacement along the y
direction when mouse is moved vertically
  var targetY = transformValue(currMousePtrLoc.y,-.75,.75,
controller.aircraftDefaultHeight-
controller.aircraftAmplitudeHt,
controller.aircraftDefaultHeight+controller.aircraftAmplitu
deHt);
  // controller.aircraftCollisionYSpeed +=
controller.aircraftCollisionYSpeed;
  targetY += controller.aircraftCollisionYSpeed;


  //displace the aircraft based on mouse movement. The
factor dT is multiplied here because the faster
  //the mouse is moved, faster should be the change in
position of the aircraft.
  //NOTE that dT is the time gap between successive
animation frames.
  aircraft.mesh.position.y += (targetY-
aircraft.mesh.position.y)*dT*controller.aircraftMovementSen
si;
  aircraft.mesh.position.x += (targetX-
aircraft.mesh.position.x)*dT*controller.aircraftMovementSen
si;


  //same for rotation of aircraft about the x axis and z
axis. need to multiple by a factor of dT
```

```
  aircraft.mesh.rotation.z = (targetY-
aircraft.mesh.position.y)*dT*controller.aircraftRotationXSe
nsi;

  aircraft.mesh.rotation.x = (aircraft.mesh.position.y-
targetY)*dT*controller.aircraftRotationZSensi;



  //move the camera along the z direction when the
aircraft's speed changes

  var targetCameraZ =
transformValue(controller.aircraftSpeed,
controller.minAircraftSpeed, controller.maxAircraftSpeed,
controller.nearPosOfCamera, controller.farPosOfCamera);

  perspCam.fov = transformValue(currMousePtrLoc.x,-1,1,40,
80);

  //call the below function for the above statement to take
effect (for the fov field to get updated)

  perspCam.updateProjectionMatrix();

  //move the camera up/down along the aircraft's movement
in y direction

  perspCam.position.y += (aircraft.mesh.position.y -
perspCam.position.y)*dT*controller.cameraSensivity;

  //increase the collision speed/displacement parameters
with the passage of every animation frame.

  controller.aircraftCollisionXSpeed += (0-
controller.aircraftCollisionXSpeed)*dT * 0.03;

  controller.aircraftCollisionXDisplacement += (0-
controller.aircraftCollisionXDisplacement)*(WIDTH/15000);

  controller.aircraftCollisionYSpeed += (0-
controller.aircraftCollisionYSpeed)*dT * 0.03;

  controller.aircraftCollisionYSpeed += (0-
controller.aircraftCollisionYSpeed)*dT *0.01;

}



/**************** 3: SKY ************ */
```

```
/* ***************Functions related to creation of the
sky.************************** */

/* This function creates the sky. */

Sky = function() {

  this.mesh = new THREE.Object3D();

  this.nClouds = 20;

  this.clouds = [];

  var stepAngle = Math.PI*2 / this.nClouds;

  for(var i=0; i<this.nClouds; i++){    // creating and
populating clouds in sky.

    var cloud = new Cloud();

    this.clouds.push(cloud);

    var ang = stepAngle*i;

    var ht = controller.radiusOfSea + Math.random()*200 +
150;

    cloud.mesh.position.x = ht*(Math.cos(ang));

    cloud.mesh.position.y = ht*(Math.sin(ang));

    cloud.mesh.rotation.z = ang + Math.PI/2;

    cloud.mesh.position.z = -300-(Math.random()*500);

    var scl = 1+(Math.random()*2);

    cloud.mesh.scale.set(scl,scl,scl);

    this.mesh.add(cloud.mesh);       // adding clouds to
sky mesh.

  }

}


/* This function creates our sky. */

function generateSky(){

  sky = new Sky();

  sky.mesh.position.y = -controller.radiusOfSea;

  scene.add(sky.mesh);
```

```
}


/* This function move clouds in sky. */
Sky.prototype.moveClouds = function(){
  for(var k=0; k<this.nClouds; k++){
    var cld = this.clouds[k];
    cld.rotate();
  }
  this.mesh.rotation.z += controller.speed*dT;
}



/*************** 4: OCEAN ************ */
/* ******************Function related to ocean
creation.******************** */
/* Function used for creating ocean. */
Ocean = function(){
  var geo = new
THREE.CylinderGeometry(controller.radiusOfSea,controller.ra
diusOfSea,controller.lengthOfSea,40,10);
  geo.applyMatrix(new
THREE.Matrix4().makeRotationX(Math.PI/2));
  geo.mergeVertices();
  var lengthOfVer = geo.vertices.length;


  this.waves = [];


  for (var i=0;i<lengthOfVer;i++){
    var vert = geo.vertices[i];
    this.waves.push({y:vert.y, x:vert.x, z:vert.z,
ang:Math.random()*Math.PI*2, amp:controller.minSizeOfWaves
+ Math.random()*(controller.maxSizeOfWaves-
```

```
controller.minSizeOfWaves),
speed:controller.minSpeedOfWaves +
Math.random()*(controller.maxSpeedOfWaves -
controller.minSpeedOfWaves)});
  };
  var matr = new THREE.MeshPhongMaterial({
    color: usefulColours.lightOceanGreen,
    transparent: true,
    opacity:.7,
    shading:THREE.FlatShading,

  });
  this.mesh = new THREE.Mesh(geo, matr);
  this.mesh.name = "waves";
  this.mesh.receiveShadow = true;
}

/* This function creates our set of clouds. */
function generateOcean(){
  ocean = new Ocean();
  ocean.mesh.position.y = -controller.radiusOfSea;
  scene.add(ocean.mesh);
}

/* Function to make and move waves in ocean */
Ocean.prototype.moveWaves = function (){
  var verts = this.mesh.geometry.vertices;
  var vertLen = verts.length;
  for (var i=0; i<vertLen; i++){
    var singleVert = verts[i];
    var vertexOfWaves = this.waves[i];
```

```
    singleVert.x =  vertexOfWaves.x +
Math.cos(vertexOfWaves.ang)*vertexOfWaves.amp;
    singleVert.y = vertexOfWaves.y +
vertexOfWaves.amp*(Math.sin(vertexOfWaves.ang));
    vertexOfWaves.ang += vertexOfWaves.speed*dT;
    this.mesh.geometry.verticesNeedUpdate=true;
  }
}



/*************** 5: CLOUDS ************ */
/* *******Functions related to creation and movement of
clouds************ */


/* Function for creating clouds.  */
Cloud = function(){
  this.mesh = new THREE.Object3D();
  this.mesh.name = "cloud";
  var geomet = new THREE.CubeGeometry(20,20,20);
  var matr = new
THREE.MeshPhongMaterial({color:usefulColours.silver,});
  var nBlocks = 3 + (Math.floor(Math.random()*3));
  for (var i=0; i<nBlocks; i++ ){
    var mesH = new THREE.Mesh(geomet.clone(), matr);
    mesH.position.x = i*15;
    mesH.position.y = Math.random()*10;
    mesH.position.z = Math.random()*10;
    mesH.rotation.y = 2*(Math.random()*Math.PI);
    mesH.rotation.z = 2*(Math.random()*Math.PI);
    var sRand = .1 + Math.random()*.9;
    mesH.scale.set(sRand,sRand,sRand);
```

```
    mesH.castShadow = true;
    mesH.receiveShadow = true;
    this.mesh.add(mesH);
  }
}

/* Functions used for rotating the clouds. */
Cloud.prototype.rotate = function(){
  var len = this.mesh.children.length;
  for(var i=0; i<len; i++){
    var mChild = this.mesh.children[i];
    //console.log(mChild);
    mChild.rotation.y+= Math.random()*(i+1)*.002;
    mChild.rotation.z+= Math.random()*(i+1)*.005;
  }
}



/*************** 6: Enemy Meteors  ************ */
/* ******************Functions used for creating enemy
meteors obstacles**************** */


/*  This function creates a single meteor obstacle. */
Meteor = function(){
  var geom = new THREE.TetrahedronGeometry(8,2);
  var matrl = new
THREE.MeshPhongMaterial({color:usefulColours.violet,
shininess:0, specular:0xffffff,shading:THREE.FlatShading});
  this.mesh = new THREE.Mesh(geom,matrl);
  this.mesh.castShadow = true;
  this.angle = 0;
```

```
  this.dist = 0;
}

/*  This function populates the meteor holder.   */
MeteorsHolder = function (){
  this.mesh = new THREE.Object3D();
  this.meteorsInUse = [];
}

/* This function creates meteors and adds them to scene. */
function generateMeteors(){
  for (var i=0; i<10; i++){
    var meteor = new Meteor();
    meteorsPool.push(meteor);
  }
  meteorsHolder = new MeteorsHolder();
  scene.add(meteorsHolder.mesh)
}

/* This function creates a set of meteor obstacles. */
MeteorsHolder.prototype.spawnMeteors = function(){
  var nmeteors = controller.stage;

  for (var i=0; i<nmeteors; i++){
    var meteor;
    if (meteorsPool.length) {
      meteor = meteorsPool.pop();
    }else{
      meteor = new Meteor();
```

```
    }


    meteor.angle = - (i*0.1);

    meteor.pointsScored = controller.aircraftDefaultHeight
+ controller.radiusOfSea +  (-1 + Math.random() * 2) *
controller.aircraftAmplitudeHt;

    meteor.mesh.position.x =
Math.cos(meteor.angle)*meteor.pointsScored;

    meteor.mesh.position.y = -(controller.radiusOfSea) +
Math.sin(meteor.angle)*meteor.pointsScored;


    this.mesh.add(meteor.mesh);

    this.meteorsInUse.push(meteor);

  }

}


/*This function rotates meteor obstacles on their place
(rotational motion). */

MeteorsHolder.prototype.rotateMeteors = function(){

  for (var i=0; i<this.meteorsInUse.length; i++){

    var meteor = this.meteorsInUse[i];

    meteor.angle +=
controller.speed*controller.meteorsSpeed*dT;


    if (meteor.angle > Math.PI*2)

        meteor.angle -= Math.PI*2;


    meteor.mesh.position.y = -(controller.radiusOfSea) +
Math.sin(meteor.angle)*meteor.pointsScored;

    meteor.mesh.position.x =
Math.cos(meteor.angle)*meteor.pointsScored;

    meteor.mesh.rotation.y += (Math.random())*.1;
```

```
    meteor.mesh.rotation.z += (Math.random())*.1;


    var diff_position =
aircraft.mesh.position.clone().sub(meteor.mesh.position.clo
ne());
    var diff = diff_position.length();
    if (diff < controller.distanceToleranceInMeteor){

fragmentsHolder.spawnFragments(meteor.mesh.position.clone()
, 15, usefulColours.violet, 3);



meteorsPool.unshift(this.meteorsInUse.splice(i,1)[0]);
      this.mesh.remove(meteor.mesh);
      controller.aircraftCollisionXSpeed = 100*
diff_position.x / diff;
      controller.aircraftCollisionYSpeed = 100 *
diff_position.y / diff;
      ambLight.intensity = 5;
      var meteor_planeSound = new
Audio('sounds/rockBreaking.wav');
      meteor_planeSound.play();
      meteor_planeSound.volume = 1;
      decreaseHealth();
      i--;
    }else if (meteor.angle > Math.PI){

meteorsPool.unshift(this.meteorsInUse.splice(i,1)[0]);
      this.mesh.remove(meteor.mesh);
      i--;
    }
  }
}
```

```
/*************** 7: METEOR FRAGMENTS  *********** */

/* **********Functions used for fragements when meteors
explode********** */

/*This function creates individual fragments of the
obstacle meteors.*/

Fragment = function(){

  var geomt = new THREE.TetrahedronGeometry(3,0);

  var matrl = new
THREE.MeshPhongMaterial({color:0x009999,shininess:0,specula
r:0xffffff,shading:THREE.FlatShading});

  this.mesh = new THREE.Mesh(geomt,matrl);

}


/* This function populates the object which holdes the
particles generated from breaking of meteors. */

FragmentsHolder = function (){

  this.mesh = new THREE.Object3D();

}


/* This function generates the fragments of the metoers
after it's destruction and add those fragements to the
scene. */

function generateFragments(){

  for (var i=0; i<10; i++){

    var fragment = new Fragment();

    fragmentsPool.push(fragment);

  }

  fragmentsHolder = new FragmentsHolder();

  scene.add(fragmentsHolder.mesh)

}
```

```
/* This function creates graphics for exploading meteor.
Here we are using TweenMax library available in Three js

   which is responsible for effects related to breaking and
movement of meteor particles after explosion.*/

Fragment.prototype.explode = function(postn, color, scale){

  var thisObject = this;

  var parentObj = this.mesh.parent;

  this.mesh.material.color = new THREE.Color(color);

  this.mesh.material.needsUpdate = true;

  this.mesh.scale.set(scale, scale, scale);

  var tgtY = postn.y + 50*(-1 + Math.random()*2);

  var tgtX = postn.x + 50*(-1 + Math.random()*2);

  var speed = .6 + (Math.random()*(.2));

  TweenMax.to(this.mesh.rotation, speed,
{x:Math.random()*12, y:Math.random()*12});   // using Twin
Max Library in Three js

  TweenMax.to(this.mesh.scale, speed, {x:.1, y:.1, z:.1});

  TweenMax.to(this.mesh.position, speed, {x:tgtX, y:tgtY,
delay:Math.random() *.1, ease:Power2.easeOut,
onComplete:function(){

      if(parentObj) parentObj.remove(thisObject.mesh);

      thisObject.mesh.scale.set(1,1,1);

      fragmentsPool.unshift(thisObject);

    }});

}


FragmentsHolder.prototype.spawnFragments = function(postn,
density, color, scale){


  var nParticles = density;

  for (var i=0; i<nParticles; i++){
```

```
    var frag;
    if (fragmentsPool.length) {
      frag = fragmentsPool.pop();
    }else{
      frag = new Fragment();
    }
    this.mesh.add(frag.mesh);
    frag.mesh.visible = true;
    var _this = this;
    frag.mesh.position.y = postn.y;
    frag.mesh.position.x = postn.x;
    frag.explode(postn,color, scale);
  }
}



/*************** 8: Jewels  *********** */
/* Functions for creating, collecting and movement of
jewels. */


/* This function creates a single jewel object and set
values of it's properties. */
Jewel = function(){
  var geomt = new THREE.TetrahedronGeometry(5,0);
  var matrl = new THREE.MeshPhongMaterial({color:
usefulColours.limeGreen,shininess:0, specular:0xffffff,
shading:THREE.FlatShading});
  this.mesh = new THREE.Mesh(geomt,matrl);
  this.mesh.castShadow = true; // shadow activated
  this.angle = 0;  // setting angle to be 0
  this.dist = 0;   // setting distance to be 0
```

```
}


/* This function function creates the given no of jewels
and populates the object which holdes the jewels.*/
JewelsHolder = function (jewels){
  this.mesh = new THREE.Object3D();
  this.jewelsInUse = [];
  this.jewelsPool = [];
  for (var k=0; k < jewels; k++){
    var jwl = new Jewel();
    this.jewelsPool.push(jwl);
  }
}


/*This function creates a set of jewels and adds them to
scene. */
function generateJewels(){


  jewelsHolder = new JewelsHolder(20);
  scene.add(jewelsHolder.mesh)
}


/* This functions is used to generate jewels. */
JewelsHolder.prototype.spawnJewels = function(){


  var jewelN = 1 + Math.floor(Math.random()*10);
  var distance_from_sea = controller.radiusOfSea +
controller.aircraftDefaultHeight + (-1 + Math.random() * 2)
* controller.aircraftAmplitudeHt;
  var amplitude = Math.round(Math.random()*10) + 10;
  for (var i=0; i<jewelN; i++){
```

```
    var jewel;
    if (this.jewelsPool.length) {
      jewel = this.jewelsPool.pop();
    }else{
      jewel = new Jewel();
    }
    this.mesh.add(jewel.mesh);
    this.jewelsInUse.push(jewel);
    jewel.angle = - (i*0.02);
    jewel.pointsScored = distance_from_sea +
Math.cos(i*.5)*amplitude;
    jewel.mesh.position.y = -controller.radiusOfSea +
Math.sin(jewel.angle)*jewel.pointsScored;
    jewel.mesh.position.x =
Math.cos(jewel.angle)*jewel.pointsScored;
  }
}
/* This functions move the current set of jewels. It also
checks whether a jewel is collected by aircraft or not. */
JewelsHolder.prototype.rotateJewels = function(){
  for (var i=0; i<this.jewelsInUse.length; i++){
    var jewel = this.jewelsInUse[i];
    if (jewel.exploding) continue;
    jewel.angle +=
controller.speed*controller.speedOfJewel*dT;
    if (jewel.angle>Math.PI*2) jewel.angle -= Math.PI*2;
    jewel.mesh.position.x =
Math.cos(jewel.angle)*jewel.pointsScored;
    jewel.mesh.position.y = -controller.radiusOfSea +
jewel.pointsScored*(Math.sin(jewel.angle));
    jewel.mesh.rotation.y += Math.random()*.1;
    jewel.mesh.rotation.z += Math.random()*.1;
```

```
    var diff_position =
aircraft.mesh.position.clone().sub(jewel.mesh.position.clon
e());
    var diff = diff_position.length();
    if (diff < controller.jewelDistanceToler){

this.jewelsPool.unshift(this.jewelsInUse.splice(i,1)[0]);
      this.mesh.remove(jewel.mesh);

fragmentsHolder.spawnFragments(jewel.mesh.position.clone(),
5, 0x009999, .8);
      increaseHealth();
      var jewelSound = new Audio("sounds/jewel.mp3");
      jewelSound.play();
      jewelSound.volume = 0.6;
      i--;
    }else if (jewel.angle > Math.PI){

this.jewelsPool.unshift(this.jewelsInUse.splice(i,1)[0]);
      this.mesh.remove(jewel.mesh);
      i--;
    }
  }
}



/*******************************************************
 * ** FUNCTIONS FOR UPDATING POINTS/HEALTH RELATED INFO ***
 * *****************************************************
 */
```

```
/* This function updates points scored in game. */

function changePoints(){

  controller.pointsScored +=
controller.speed*dT*controller.ratioOfSpeedDistance;

  currPoints.innerHTML =
Math.floor(controller.pointsScored);

  var incrementPoints = 502*(1-
(controller.pointsScored%controller.distanceForStageUpdate)
/controller.distanceForStageUpdate);

  stageContainer.setAttribute("stroke-dashoffset",
incrementPoints);

}


/* This function updates health of the aircraft. */

function changeHealth(){

  controller.health -=
controller.speed*dT*controller.ratioOfSpeedEnergy;

  controller.health = Math.max(0, controller.health); //
updating health value.

  healthMeterFill.style.right = (100-
controller.health)+"%"; //stylng health meter.

  //updating bg colour for health.

  if(controller.health<50){

    healthMeterFill.style.backgroundColor =  "#00ffff";

  }

  else{

    healthMeterFill.style.backgroundColor = "#66cdaa";

  }

  if (controller.health<30){

    healthMeterFill.style.backgroundColor = "#f25346";

    healthMeterFill.style.animationName = "flickering";

  }else{
```

```
    healthMeterFill.style.animationName = "none";
  }
  // updating health when game is over.
  if (controller.health <1){
    controller.stateOfGame = "gameIsOver";
    var gOver = new Audio("sounds/gameOver.mp3");
    gOver.play();
    gOver.volume = 1;
  }
}


/* This function increments the health of the aircraft when
it collects points. */
function increaseHealth(){
  controller.health += controller.valueOfJewel;
  controller.health = Math.min(controller.health, 100);
}


/* This function decreases health of the aircraft when it
hits any meteor obstacle. */
function decreaseHealth(){
  controller.health -= controller.healthLossByMeteor;
  controller.health = Math.max(0, controller.health);
  if (controller.health <1){
    controller.stateOfGame = "gameIsOver";
    var gOver = new Audio("sounds/gameOver.mp3");
    gOver.play();
    gOver.volume = 1;
  }
}
```

```
/**
 *
************************************************************
****************************************
 * The following 4 functions are used for showing messages
on the screen at different states of the game.
 *
************************************************************
****************************************
 * */
function showReplay(){
  //blocking display
  replayMessage.style.display="block";
}
function hideReplay(){
  replayMessage.style.display="none";
}
function showStart(){
  //blocking display
  startMessage.style.display="block";
}
function hideStart(){
  startMessage.style.display="none";
}


/* This function normalizes/transforms the value 'value'
(that lies in range minValue,maxValue)
  into a value that lies in the target range (minTarget,
maxTarget) */
function transformValue(value,minValue,maxValue,minTarget,
maxTarget){
```

```
  var newValue = Math.max(Math.min(value,maxValue),
minValue);  //normalizing the X/Y position of mouse by
constraining it between vmin and vmax

  var rangeValues = maxValue-minValue;

  var positionChange = (newValue-minValue)/rangeValues;

  var targetChange = maxTarget-minTarget;

  var transformedValue = minTarget +
(positionChange*targetChange);

  return transformedValue;

}



/************* Standard function that initializes the game,
renders objects and create animation frames.

 It is actually starting point of the whole game.
***********/

function init(event){


  //user interface

  startMessage =
document.getElementById("clickToStartMsg");

  currPoints = document.getElementById("distValue");

  healthMeterFill =
document.getElementById("healthMeterFill");

  replayMessage =
document.getElementById("clickToReplayMsg");

  stageField = document.getElementById("currStage");

  bulletVal = document.getElementById("bulletCount");

  stageContainer =
document.getElementById("stageContainerStroke");



  //event listeners for various events on the document
```

```
  document.addEventListener('mousemove',
mousePtrMoveHandler, false);

  document.addEventListener('mouseup',
mouseClickReleaseHandler, false);

  document.addEventListener('keyup',
keyButtonReleaseHandler, false);

  document.addEventListener('keydown',
keyButtonPressHandler, false);


  //restart the game

  generateNewGame();

  //set the scene

  generateScene();

  //can now add window resize handler since scene is
created

  window.addEventListener('resize',
windowSizeChangeHandler, false);

  //set the lighting

  generateLighting();


  //make the aircraft

  generateAircraft();

  //make the ocean

  generateOcean();

  //make the sky

  generateSky();

  //make the jewels

  generateJewels();

  //make the meteors/obstacles

  generateMeteors();

  //make the meteor fragments

  generateFragments();
```

```
  //updating the animation frame.
  loop();

}
window.addEventListener('load', init, false);




/* This function get executed at the beginning of each
animation frame.
   In this function all objects (such as jewles, aircraft
etc) gets updated
    depending upon the current state of the game. */
function loop(){


  currFrameTime = new Date().getTime();
  dT = currFrameTime-prevFrameTime;
  prevFrameTime = currFrameTime;
  // checking current state of the game and performing
actions depending upon them.


  if(controller.stateOfGame == "notStarted"){
      showStart();
      controller.stateOfGame = "waitingForStart";
  }
  else if(controller.stateOfGame == "waitingForStart"){


  }
  else if (controller.stateOfGame=="currentlyInGame"){
    // Add health jewels for every 100m;
```

```
    if
(Math.floor(controller.pointsScored)%controller.jewelSpawnD
istance == 0 && Math.floor(controller.pointsScored) >
controller.lastSpawnOfJewel){

      controller.lastSpawnOfJewel =
Math.floor(controller.pointsScored);

      jewelsHolder.spawnJewels();

    }


    // Increase speed to the aircraft

    if
(Math.floor(controller.pointsScored)%controller.distToUpdat
eSpeed == 0 && Math.floor(controller.pointsScored) >
controller.lastUpdatedSpeed){

      controller.lastUpdatedSpeed =
Math.floor(controller.pointsScored);

      controller.targetBaseSpeed +=
controller.increaseSpeedWithTime*dT;

    }


    // Add obstacles meteor for every 100m.

    if
(Math.floor(controller.pointsScored)%controller.meteorSpawn
Distance == 0 && Math.floor(controller.pointsScored) >
controller.lastSpawnOfMeteor){

      controller.lastSpawnOfMeteor =
Math.floor(controller.pointsScored);

      meteorsHolder.spawnMeteors();

    }


    // increment stage variable and sound for level up.

    if
(Math.floor(controller.pointsScored)%controller.distanceFor
StageUpdate == 0 && Math.floor(controller.pointsScored) >
controller.stageLastUpdate){
```

```
      controller.stageLastUpdate =
Math.floor(controller.pointsScored);

      controller.stage++;

      stageField.innerHTML = Math.floor(controller.stage);

      controller.bulletCount = controller.stage * 2;

      bulletVal.innerHTML =
Math.floor(controller.bulletCount);

      controller.targetBaseSpeed = controller.startSpeed +
controller.increaseSpeedWithLevel*controller.stage;

      var levelUp = new Audio("sounds/levelUp.mp3");

      levelUp.play();

      levelUp.volume = 1;


    }



    updateAircraft();

    changePoints();

    changeHealth();

    updateBullet();


    // increasing speed of aircraft.

    controller.baseSpeed += (controller.targetBaseSpeed -
controller.baseSpeed)* 0.02 * dT;

    controller.speed = controller.baseSpeed *
controller.aircraftSpeed;


  }else if(controller.stateOfGame=="gameIsOver"){   // when
game gets over show message and await for restart.

    controller.speed *= .99;

    aircraft.mesh.rotation.z += (-Math.PI/2 -
aircraft.mesh.rotation.z)*.0002*dT;
```

```
    aircraft.mesh.rotation.x += 0.0003*dT;
    controller.aircraftFallSpeed *= 1.05;
    aircraft.mesh.position.y -=
controller.aircraftFallSpeed*dT;

    if (aircraft.mesh.position.y <-200){
      showReplay();
      controller.stateOfGame = "awaitingGameRestart";

    }
  }else if (controller.stateOfGame=="awaitingGameRestart"){

  }

  // updating aircraft rotation.
  aircraft.rotor.rotation.x +=.2 + controller.aircraftSpeed
* dT*.005;

  // updating ocean.
  ocean.mesh.rotation.z += controller.speed*dT;

  if ( ocean.mesh.rotation.z > 2*Math.PI)
      ocean.mesh.rotation.z -= 2*Math.PI;

  ambLight.intensity += (.5 - ambLight.intensity)*dT*0.005;

  // moving jewels.
  jewelsHolder.rotateJewels();
  meteorsHolder.rotateMeteors();
```

```
  // moving environment variables
  sky.moveClouds();
  ocean.moveWaves();

  //rendering scene in webgl using prepective projection
camera.
  webGlRenderer.render(scene, perspCam);
  requestAnimationFrame(loop);
}




/* *
 * FUNCTIONS FOR HANDLING ON SCREEN MOUSE/KEYBOARD EVENTS
 * These functions are passed as callbacks to the event
listeners
*/


/* Handles resizing of the window  */
 function windowSizeChangeHandler() {
  webGlRenderer.setSize(WIDTH, HEIGHT);
  perspCam.aspect = WIDTH / HEIGHT;
  //need to call the below function for the
  //attribute changes to take effect
  perspCam.updateProjectionMatrix();
}


/* Handles movement of mouse pointer  */
function mousePtrMoveHandler(){
  var adjustedX = (event.clientX / WIDTH)*2;
```

```
  adjustedX--;
  var adjustedY = (event.clientY / HEIGHT)*(-2);
  adjustedY++;
  currMousePtrLoc = {x:adjustedX, y:adjustedY};
}

/* Handles release of mouse click */
function mouseClickReleaseHandler(event){
  if (controller.stateOfGame == "awaitingGameRestart"){
    hideReplay();
    generateNewGame();
  }
  else if(controller.stateOfGame == "waitingForStart"){
    hideStart();
    generateNewGame();
    var bgAudio = document.getElementById("backAudio");
    if(bgAudio.canPlayType('audio/mp3')){
      bgAudio.setAttribute('src','sounds/background.mp3');
      console.log("WORKING");
      bgAudio.play();
      //oye..vs code on he na
      bgAudio.volume = 0.5;
    }
    controller.stateOfGame = "currentlyInGame";
  }
}

/* Handles pressing of a button on the keyboard  */
function keyButtonPressHandler(event){
```

```
  if (controller.stateOfGame == "awaitingGameRestart"){
    generateNewGame();
    controller.stateOfGame = "currentlyInGame";
    hideReplay();
  }
  else if(controller.stateOfGame == "waitingForStart"){
    hideStart();
    generateNewGame();
    var bgAudio = document.getElementById("backAudio");
    if(bgAudio.canPlayType('audio/mp3')){
      bgAudio.setAttribute('src','sounds/background.mp3');
      console.log("WORKING");
      bgAudio.play();
      //oye..vs code on he na
      bgAudio.volume = 0.5;
    }
    controller.stateOfGame = "currentlyInGame";
  }
  else if(controller.stateOfGame == "currentlyInGame"){
    if(canFireBullet == 1 && event.key == "x"){
      generateBullet();
      //console.log("\nheree");
      canFireBullet = 0;
    }
  }
}

/* Handles releasing of a button on the keyboard  */
function keyButtonReleaseHandler(event){
```

```
  if(event.key == "x"){
    canFireBullet = 1;
  }
}
```

MARIO JUMP

1. HTML

```
<!DOCTYPE html>

<html lang="pt-BR">

<head>

    <!--Meta Settings-->

    <meta charset="UTF-8">

    <meta http-equiv="X-UA-Compatible" content="IE=edge">

    <meta name="viewport" content="width=device-width,
initial-scale=1.0">


    <!--Favicon-->
```

```
    <link rel="shortcut icon" href="img/favicon.png"
type="image/x-icon">


    <!--Link Stylesheet-->
    <link rel="stylesheet" href="css/style.css">


    <!--Title-->
    <title>Mario Jump</title>
</head>
<body>
    <!--Game-->
    <audio src="audio/jump_song.mp3"
class="audiojump"></audio>
    <audio src="audio/game_over.mp3"
class="gameover"></audio>
    <div class="game-board">
        <img src="img/clouds.png" class="clouds">
        <img src="img/Tatooine.png" class="mario">
        <img src="img/pipe.png" class="pipe">
    </div>
    <p id="text-start">Para jogar, pressione qualquer
tecla</p>
    <!--Script-->
    <script src="js/script.js"
onclick="rotinapular()"></script>
</body>
</html>
```

2. CSS

```
/*==================Global
Settings====================*/
* {
   margin: 0;
```

```
    padding: 0;
    box-sizing: border-box;
    user-select: none;
}

p {
    font-family: 'Press Start 2P', monospace;
    padding-top: 1.5%;
    text-align: center;
    color: black;
}
/*=========================================================
============*/

/*====================Game
Settings====================*/
.game-board {
    width: 100%;
    height: 90vh;
    border-bottom: 15px solid green;
    margin: 0 auto;
    position: relative;
    overflow: hidden;
    background: linear-gradient(#87CEEB, #E0F6FF) ;
}

.pipe {
    position: absolute;
    bottom: 0;
```

```
    width: 80px;
    animation: pipe-animation 1.5s infinite linear;
}

.mario {
    width: 150px;
    position: absolute;
    bottom: 0;
}

.jump {
    animation: jump 500ms ease-out;
}

.clouds {
    position: absolute;
    width: 550px;
    animation: clouds-animation 20s infinite linear;
}




@keyframes pipe-animation {
    from {
        right: -80px;
    }
    to {
        right: 100%;
```

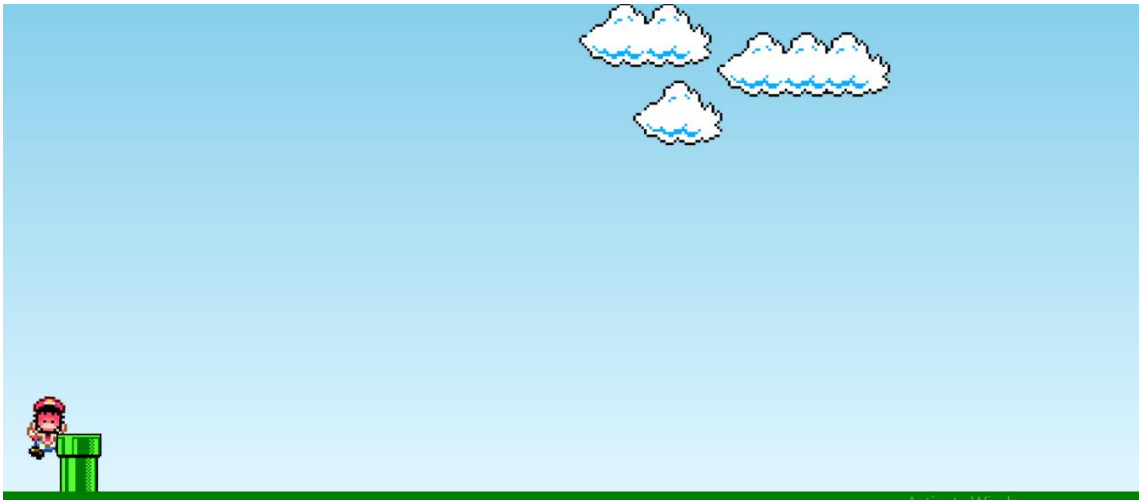
```
    }
}

@keyframes jump {
    0% {
        bottom: 0;
    }
    40% {
        bottom: 180px;
    }
    50% {
        bottom: 180px;
    }
    60% {
        bottom: 180px;
    }
    100% {
        bottom: 0;
    }
}

@keyframes clouds-animation {
    from {
        right: -550px;
    }
    to {
        right: 100%;
    }
```

```
}
```

3. JS

```
/* = associar a uma constante os seletores CSS */

const mario = document.querySelector('.mario');

const pipe = document.querySelector('.pipe');

const audioJump = document.querySelector('.audiojump');

const gameOver = document.querySelector('.gameover');

const textStart = document.querySelector('#text-start')




/*==========================loop principal do
Game==========================*/

const loop = setInterval(() => {

    const pipePosition = pipe.offsetLeft;

    const marioPosition = +window.getComputedStyle(mario).bottom.replace('px',
'');


    /* condição lógica que define COLISÃO */

    if (pipePosition <= 120 && pipePosition > 0 && marioPosition < 80) {

        pipe.style.animation = 'none';

        pipe.style.left = `${pipePosition}px`;


        mario.style.animation = 'none';

        mario.style.bottom = `${marioPosition}px`;


        mario.src = 'img/game-over.png';

        mario.style.width = '75px';

        mario.style.marginLeft = '50px'
```

```
        /* Disparo o som do Game over ,
        usando os atributos css da constante gme over */
        gameOver.currentTime = 0.1;
        gameOver.volume = 0.2;
        gameOver.play();


        document.getElementById("text-start").style.color = "black";
        document.getElementById("text-start").innerHTML = "<strong>GAME
OVER</strong>";
        clearInterval(loop);
    }
}, 10);


const rotinapular = () => {
    mario.classList.add('jump');
    audioJump.currentTime = 0.1;
    audioJump.volume = 0.1;
    audioJump.play();
    setTimeout(() => {
        mario.classList.remove('jump');
    }, 500);
}
document.addEventListener('keydown', rotinapular);
```



CAR GAME

1. HTML

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width, initial-
scale=1.0">

    <link rel="icon" href="./title-img.png">
    <link rel="stylesheet" href="style.css">
    <title>Car Animation</title>
</head>
<body>
    <h1 class="typing">
        Car Animation
    </h1>
    <!-- <h3>&#169; 2021 Saim. All rights reserved</h3> -->
    <div class="night">
        <h2>
```

```
            <p>Press [Enter] to move the car</p>
            <p>Press [W] to turn the lights on</p>
        </h2>
        <div class="surface"></div>
        <div class="car">
            <img src="./Car 1.PNG" alt="Car">
        </div>
    </div>

    <script src="https://code.jquery.com/jquery-
3.6.0.min.js"></script>
    <script src="script.js"></script>
</body>
</html>
```

2. CSS

```
@import
url('https://fonts.googleapis.com/css2?family=Poppins:wght@200;3
00;400&display=swap');
*{
    font-family: 'Poppins', sans-serif;
}
body {
    margin: 0;
    background-image: linear-gradient(to top, #2b3b97 0%,
#75b0dd 100%);
    overflow-y: hidden;
}
h1{
    position: absolute;
    top: 1px;
    left: 38%;
    font-family: Poppins;
```

```
    font-size: 2.2rem;
    font-weight: 400;
    color: beige;
    text-transform: uppercase;
    word-spacing: 5px;
    letter-spacing: 2px;
}
.typing::after{
    content: "";
    position: absolute;
    margin-top: .45rem;
    width: 1px;
    height: 2.3rem;
    border-right: 2px solid white;
    animation: blink 0.5s infinite ease;
}
h2{
    font-family: Poppins;
    font-size: 1rem;
    padding: 5px;
    margin-left: 1.5rem;
    margin-top: .5rem;
    font-weight: 300;
    color: beige;
}

/* h3{
    position: absolute;
    color: beige;
    left: auto;
    right: auto;
```

```
    font-size: 0.8rem;
    font-weight: 300;
    z-index: 1;
} */

/* Car and Scenery */

.night {
    height: 83vh;
    width: 73vw;
    margin: 6rem auto;
    background: url(Scenery\ 1.jpg);
    background-size: cover;
    position: relative;
    box-shadow: 1px 2px 60px rgba(0, 0, 0, 0.4);
    overflow-x: hidden;
}

.surface{
    height: 200px;
    width: 500%;
    background: url(Mountain\ 1.png);
    display: block;
    position: absolute;
    bottom: 0%;
    left: 0%;
    background-repeat: repeat-x;
}
.car{
    position: absolute;
    bottom: 2%;
```

```
    left: 24%;
}

.moveRight {
    animation: moveRight 6s linear infinite;
}
.suspension {
    animation: suspension 1s linear infinite;
}
/* Keyframes */

@keyframes moveRight{
    100%{transform: translateX(-2950px)}
}

@keyframes suspension{
    100%{
        transform: translateY(-1px);
    }
    50%{
        transform: translateY(2px);
    }
    0%{
        transform: translateY(-1px);
    }
}
@keyframes blink {
    0%{
        opacity: 0;
    }
    100%{
```

```
        opacity: 1;

    }

}
```

3. JS

```
$(document).ready(function(){
    //Variables
    $surface =$('.surface');
    $car =$('.car');
    $img = $('.car img');
    let flag = true;

const cars = ['./Car 1.PNG','./Car 2.PNG']

        //keypress event
    $(document).on('keypress',function(e){
        if(e.which == 13){
            $($surface).toggleClass('moveRight');
            $($car).toggleClass('suspension');
        }
    })
    $(document).on('keypress',function(e){
        if(e.which == 119){
            if(flag){
                flag = false;
                $img.attr('src', cars[0]);
            }else{
                flag = true;
                $img.attr('src', cars[1]);
            }
        }
    })
});
// Typewriting Effect //
const text = ['car animation', 'car animation', 'car animation']
```

```
let count = 0;
let index = 0;
let currentText = '';
let letter = '';
(function type(){
    if(count === text.length){
        count = 0;
    }
    currentText = text[count];
    letter = currentText.slice(0, ++index);

    document.querySelector('.typing').textContent = letter;
    if(letter.length === currentText.length){
        count++;
        index = 0;
    }
    setTimeout(type, 400);
}());
```