Crazy Air Flight

July 6, 2018

1 User manual

"Crazy Air Flight" is a simple WASD game in which the player have to move an airplane in order to avoid the impact with "strange" obstacles.

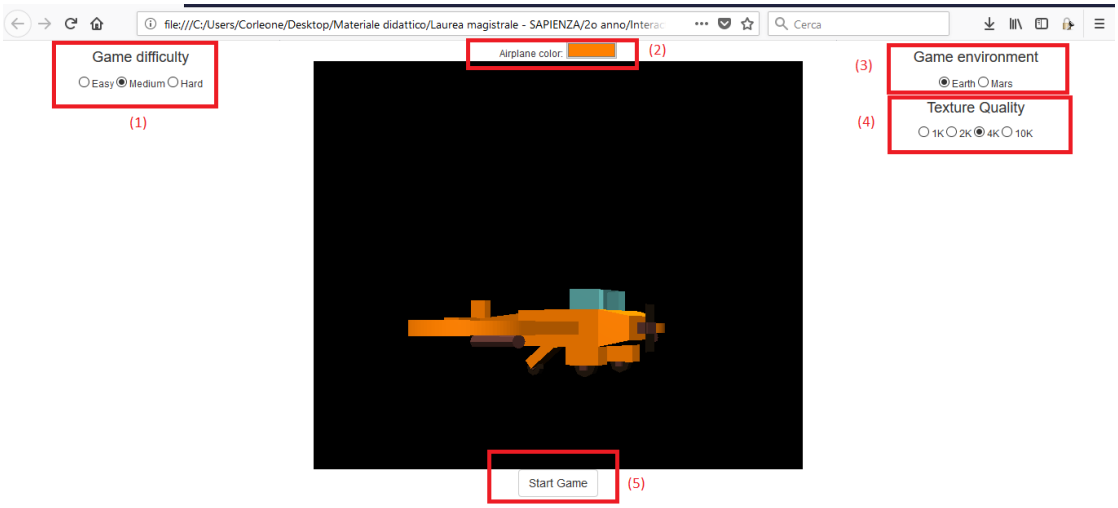

Figure 1: Initial page

In the initial page the user must select:

1. game difficulty: there are three level (easy, medium and hard). The level of the difficulty increases the speed of the obstacles.
2. airplane color
3. environment: the airplane can fly above the Earth or above Mars.

4. texture quality: the user can choose the quality of the texture. High resolution textures provide a better quality but require more computational power.

Afther selecting all this options, the user can start the game chicking the "Start" button (5).

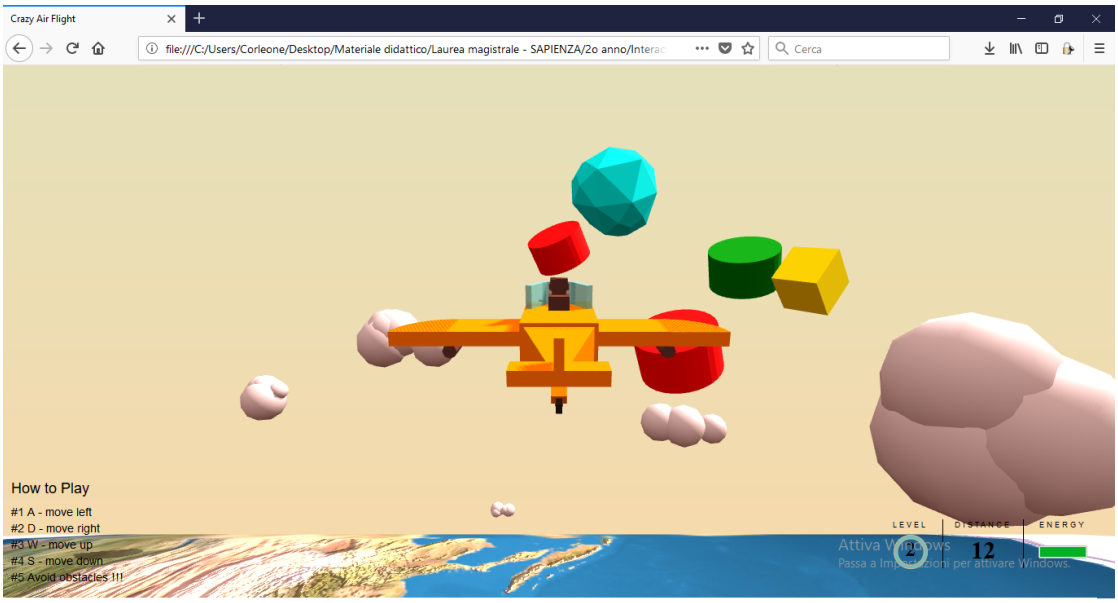

Figure 2: Initial page

At this point the user can start to play moving the airplane with the "W" (up),"A" (left),"S" (down) and "D" (right) keyboard keys.

2 Description of the used environment

Three.js is a JavaScript library for rendering interactive 2D and 3D graphics inside an HTML <canvas> element. It is based on WebGL, a lower level library which allow browsers to exploit the computational power made available by the GPU to create graphics and animations. The need to access graphic hardware derives from the fact that CPUs are not fast enough to do the huge number of calculations generally needed in realtime 3D graphics. GPUs instead are ad-hoc designed to deal with these graphics calculations.
Differently from WebGL, Three.js allow to deal with graphics according to the Object Oriented programming paradigm (O.O.P.). The possibility to use the synthetic-camera

model with O.O.P. simplify the creation of objects and complex scenarios, giving however the possibility to the programmer to customize the scene as he likes.
Among the various functionalities made available by Three.js we remember:

- basic geometric transformations (translation, scaling and rotation)
- known built-in geometries like cube, sphere and cylinder
- materials with different shading and shininess properties
- textures of different kinds (color, normal, specular, ...)
- directional, spot , ambient lights
- possibility to load 3^{th} party models
- raycasting to compute intersection

3 List of 3^{th} party resources used in the project

- GreenSock Animation Platform (GSAP) : a robust and high-performant library for HTML5 animation. It had been used only a module of the library called TweenMax for a single kind of animations.
- Earth high resolution textures from planetpixelemporium.com
- Mars high resolution textures from planetpixelemporium.com

4 Technical aspects

4.1 Models

4.1.1 Planet

The planet upon which the airplane flies is implemented as a mesh whose geometry is a Cylinder with radius 600 and length 1200. The material instead is a so called "MeshPhongMaterial" and on it are mapped different textures (map, specularMap, lightMap, normalMap).

```
.....
var geom = new THREE.CylinderGeometry(game.pianetaRadius,game.
    pianetaRadius,game.pianetaLength,40,10);
var mat = new THREE.MeshPhongMaterial({
  map: new THREE.TextureLoader(manager).load(map_path),
  bumpMap : new THREE.TextureLoader(manager).load(bump_path),
  bumpScale: 50,
  specularMap: my_spec,
  lightMap: my_light,
  normalMap: my_normal,
```

```
      specular: 0x222222,
      shininess: 25
    });
    this.mesh = new THREE.Mesh(geom, mat);
    .....
```

Listing 1: Planet model

4.1.2 Clouds

When the user chooses as environment "Earth", the scene presents also many clouds. The model that makes this possible is called "Nuvola". It creates the cloud effect generating between 8 and 11 Icosahedrons with random scale and random position but however concentred nearest a common center.

```
var Nuvola = function(){
  this.mesh = new THREE.Object3D();
  var geom = new THREE.IcosahedronGeometry(15,1);
  var mat = new THREE.MeshPhongMaterial({
    color:colori.white,
  });
  var nAtomi = 8+Math.floor(Math.random()*3);
  for (var i=0; i<nAtomi; i++ ){
    var m = new THREE.Mesh(geom, mat);
    m.position.x = Math.random()*i;
    m.position.y = Math.random()*10;
    m.position.z = Math.random()*i*4;
    m.rotation.z = Math.random()*Math.PI*2;
    m.rotation.y = Math.random()*Math.PI*2;
    var s = .1 + Math.random()*.9;
    m.scale.set(s,s,s);
    m.castShadow = true;
    m.receiveShadow = true;
    this.mesh.add(m);
  }
}
```

Listing 2: Cloud model

The entire Sky is then generated as a set of 20 clouds with random position.

4.1.3 Airplane

The principal hierarchical model used inside the project is an Airplane. It was created by scratch using only cubes modified properly to give to the model a realistic shape.

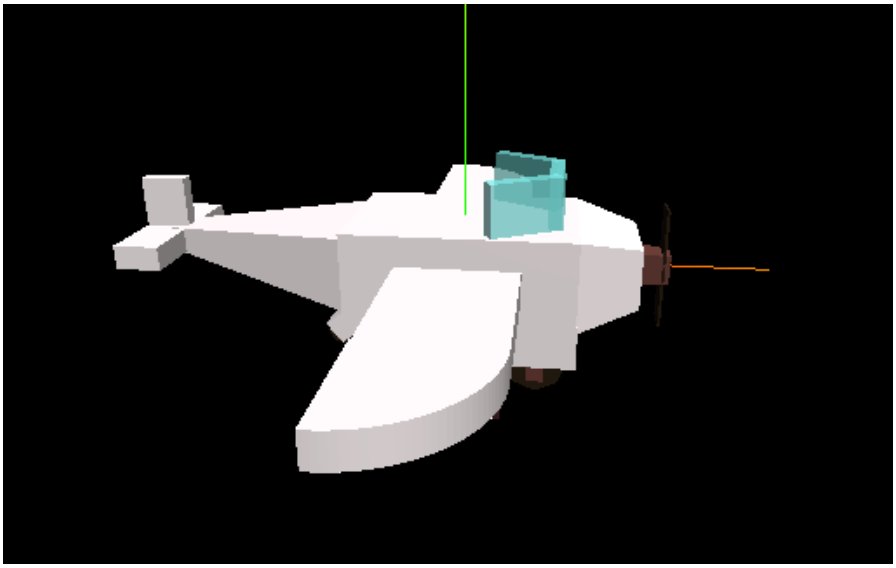

Figure 3: Aereo model

Its main components are

- tail
- the two wings
- engine
- cabin
- glass that protect the driver

The parent of the whole model is the "cabin" element. All the other elements are properly connected to it (they are its children).

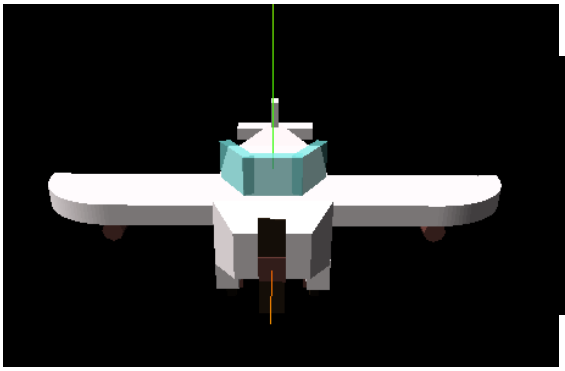

Figure 4: Front part

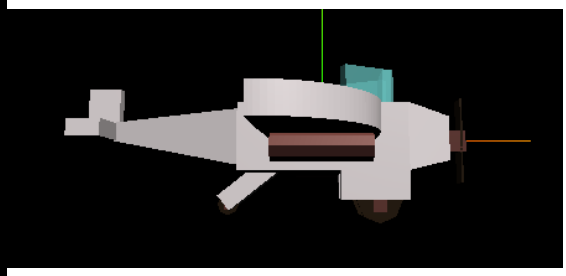

Figure 5: Bottom part

At the beginning of the game to the airplane model is attached as child the body of a human-like character called "Pilota".

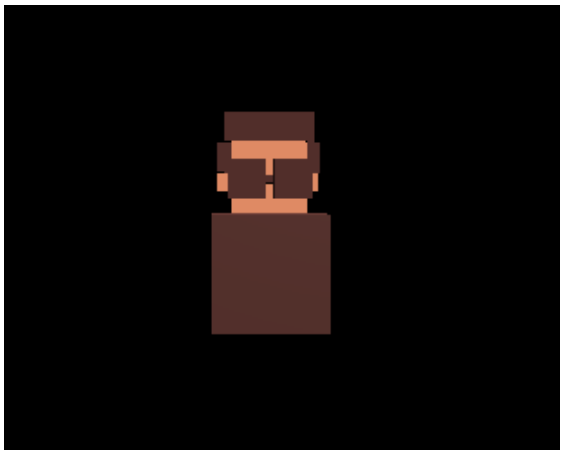

Figure 6: Pilota model

4.1.4 Obstacols

During the game the airplane is attacked by many "obstacles". The obstacles are meshes with well-known geometries : TetrahedronGeometry, SphereGeometry, BoxGeometry and CylinderGeometry. Both the geometries and the colors are choosed randomly.
Obstacles are managed through an ad hoc JavaScript object called "ContenitoreOstacoli". ContenitoreOstacoli allow to generate, remove and move obstacles during the

game. During each moment there are six obstacols arranged in the scene. The position of the obstacles is chosed randomly but such that to hit the airplane in the average case. When an obstacle hits the airplane, the last one is subject to an animation according to the direction of the impact whereas the obstacle is "dematerialized" and then deleted from the scene. Instead when an obstacle exceeds the position of the airplane, it is insert into a data structure canned "riservaOstacoli" and reused in the next generation step. This is done to avoid the filling of the memory with many objects.

4.1.5 Racimolo

It is a small mesh with TetrahedronGeometry. A set of such models are used during the impact between the airplane and the obstacols in order to simulate the dematerialization of the obstables. The dematerialization is simulated through a set of such models which are scattered around the impact position. Afther a certain time they are deleted from the scene.

4.2 Lights

The environment is illuminated by two main lights: an AmbientLight and a HierarchicalLight. The first one globally illuminates all objects in the scene equally. The second one is positioned directly above the scene, with color fading from the sky color to the ground color. During the impact between the airplane and the object the ambient light is increased of a factor of 3 and then regularly decreased.
All the objects are able to be shaded (receiveShadows) or to generate shadows (castShadows). The shadows are visilble expecially on the planet model.

4.3 Animations

During the game there are some elements of the scane which are sumbitted to constant animations.

- the planet is rotated along the z-axis of .005 rad
- the sky is rotated along the z-axis of .01 rad
- the hair of the pilot are "moved" to simulate the effect of the air current over them. Basically the hairs are a set of rectangle and the animation is achived scaling their height randomly for a factor between 0.75 and 1.
- the wheels of the airplane are rotated around the y-axis of 0.1 rad.
- the propeller of the airplane are rotated around the x-axis of 0.3 rad.

Each time there is a collision between the airplane and an obstacle, the obstacle is dematerialized and the airplane is moved in the same direction of the impact but with opposite verso. The dematerialization animation is realized with GSAP library and it is composed by three elementary animations applied to a set of racimiolo models. The

three animations basically toghether scale the models, rotate them and at the same time move them to predefined positions. The impact of the airplane with an ostacle also move it in the opposite direction. This was implemented through the usage of an ad-hoc class called Animation. Each instance of the Animation class is composed by a list of states (clips), a pointer to the current clip and a target mesh. When a such animation is fired, all the states are applied sequentially to the target mesh. All the above animations are stored into a list called "animazioni" and their "clips" are executed according to a Round-Robin policy. When the execution of an animation finish, it is deleted from the "animazioni" list.

```
function loop(){
...
...
...
  if(animazioni.length>0){
      var l=[];
      for(var i=0;i<animazioni.length;i++){
        var flag=animazioni[i].execute();
        if(!flag){
          l.push(i);
        }
      }
      for(var i=l.length-1;i>=0;i--){
        animazioni.splice(l[i],1);
      }
  }
...
...
...
}
```

Listing 3: Animazioni management

Last but not least , there is an animation that is performed each time all the energy is consumed. In this case the airplane is moved along the x-axis for a certain distance and then it is "exploded" with the same logic of the obstacles dematerialization.

5 Implemented interaction

The user can chose in the initial page the game difficulty, the environemnt, the quality of the textres and the color of the airplane. The first three features are implemented through radio buttons families. For each family, the user selects the desired option.
The airplane color is selected using an HTML color picker. Clicking on the button "Start Game", all this information are transfered to the game which i customized properly.
During the game the user interacts with the airplane through the keyboard keys "A","W","S" and "D". Each time the game is started, the previous keyboard keys can be used to move the airplane respectively on left, up, down and right. The interaction was implemented managing appropriately for each keyboard key two events: "keyup" and "keydown".

The role of the event handling methods is only to change the value of a boolean flag from true to false or viceversa.

```
document.addEventListener('keydown', function(event) {
  if(event.code == 'KeyA'){
    if(!flagD && !animazione){
      flagA=true;
    }
  }
  else if (event.code == 'KeyD') {
    if(!flagA && !animazione){
      flagD=true;
    }
  }
  else if(event.code == 'KeyW'){
    if(!flagS && !animazione){
      flagW=true;
    }
  }
  else if(event.code == 'KeyS'){
    if(!flagW && !animazione){
      flagS=true;
    }
  }
});
```

Listing 4: "keydown" event handler

This was done in order to give the possibility to the methods that handle the events to be always ready to capture the new events generated by the users. The task to manage the event was delegated to the upadateAereo method which change the inclination and the position of the airplane in an appropriate manner.

```
  if(!animazione){
    if(flagA&&(aereo.mesh.position.z-zVelocita>=-100)){
    aereo.mesh.position.z -= zVelocita;
    if(aereo.mesh.rotation.x==0) {
      aereo.mesh.rotation.x-=Math.PI/5;
    }
    }
    else if(flagD&&(aereo.mesh.position.z+zVelocita<=100)){
        aereo.mesh.position.z += zVelocita;
      if(aereo.mesh.rotation.x==0){
        aereo.mesh.rotation.x+=Math.PI/5;
      }
    }
    else if(flagW&&(aereo.mesh.position.y+yVelocita<=altezza_aereo+50)){
      aereo.mesh.position.y += yVelocita;
      if(aereo.mesh.rotation.z==0){
        aereo.mesh.rotation.z+=Math.PI/8;
      }
    }
    else if(flagS&&(aereo.mesh.position.y-yVelocita>=altezza_aereo-50)){
      aereo.mesh.position.y -= yVelocita;
```

```
      if(aereo.mesh.rotation.z==0){
        aereo.mesh.rotation.z-=Math.PI/8;
      }
    }
  }
```

Listing 5: updateAereo method