Sapienza University

SAPIENZA
UNIVERSITÀ DI ROMA

Interactive Graphics

Project report

Author:
Ekin Mehmet Şenler

January 15, 2019

Contents

1 Introduction

For the course of interactive graphic, I decided to developed WEBGL third person game. The main character is spider, the goal is to eat specific number of insect in a limited span of time. The Character can walk, run and rotate, while hunting as many insect as it can, it also has to be considered that running consume stamina of the character. In the scene, the character is placed on the terrain, insect and tree objects are distributed randomly on the map. If character had eaten sufficient number of insect in a given period, spider win the game and celebrate, otherwise spider die and game is over. At the end, two different overlay appear according to the result of the game. Both overlay has the final score and a replay button.

2 Environment

Project is constructed with **Three.js** which is an open source cross platform library to display animated 3D computer graphics in a web browser. It is very powerful and helps user to skip low level WEBGL related adjustments. Thanks to its wide range of API options, it is possible to author complex 3D computer animations that displays in the browser without the effort required for a traditional standalone application. Three.js allows the creation of Graphical Processing Unit (GPU)-accelerated 3D animations using the JavaScript language as part of a website without relying on proprietary browser plugins. This is possible due to the advent of WebGL.

3 Gameplay and user interaction

Game starts with a spider centered on the camera. Trees and insects are randomly distributed on the map. Spider is controlled with keyboard and third person camera is attached to spider's backside to follow its movements. Spider can move forward and backward and it can rotate, it also perform those movement faster.

Figure 1: Scene

Spider's stamina, which is indicated in the lower left side corner of the screen is also running out when faster movement performed, therefore to regenerate stamina, whether spider need to walk or red balls through out the map needs to be collected.

User can change the position of the Spider with different button:

W move forward

S move backward

A rotate left

D rotate right

Shift 2x faster

4 Code structure

To separate as much as possible each component, the various part of the application shouldn't know much about each other and, when possible, nothing at all.

1. Web-oriented code (relative to the DOM, events, etc), shouldn't know anything about the Three.js world and, vice versa, Three.js code shouldn't know anything about the DOM.

2. The Three.js side of the application should be modularized, it should be the composition of bunch of independent component.

A high level component (or container) is used as the entry point of the Three.js application, it is responsible for initializing and managing the scene, but it doesn't know anything about the actual content of the scene. The only thing it knows is that it contains a bunch of other components that should be updated at every frame.

Each entity of the 3D scene should have its own component and all the logic for that entity should be contained inside that component. Obviously, if needed, each entity can be a container itself and contain multiple other entities. Each entity doesn't know anything about its container.

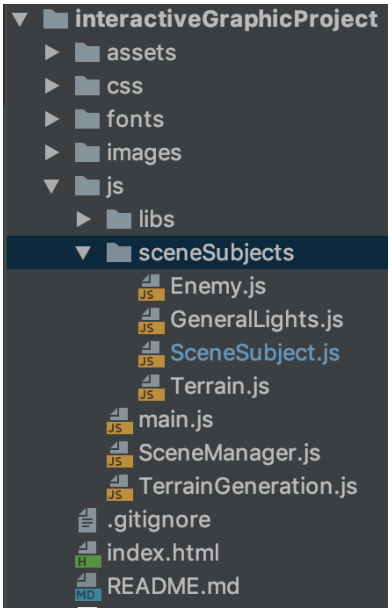


Figure 2: Project structure

Let's talk about the main and the SceneManager, The main is the entry point to the Javascript side of the application, it has access to the DOM and contains the SceneManager. The SceneManager is responsible for handling the Three.js side of the app, which is completely hidden from the main. It knows nothing about the DOM. The main has three basic responsibilities:

1. create the SceneManager, while passing a canvas to it (so that SceneManager won't have to meddle with the DOM).
2. attach listeners to the DOM events we care about (such as windowresize or mousemove).
3. start the render loop, by calling requestAnimationFrame().

On the Three.js side of the app there are two simple high level concepts:

- create Scene, Renderer and Camera.

- initialize a bunch of SceneSubjects.
- update everything at every frame.

SceneManager: SceneManager has four public methods that are called by the main: update(), onDocumentKeyDown(), onDocumentKeyUp() and onWindowResize(). Scene manager is responsible to listen to DOM event created by main.
SceneManager.update() calls:

- the *update()* method of every *SceneSubject* contained in the *SceneManager.*
- the *render()* method of the Three.js *Renderer.*

It called by main at every frame

```
this.update = function() {
   for(let i=0; i<sceneSubjects.length; i++)
        sceneSubjects[i].update();
      renderer.render(scene, camera);
}
```

SceneManager.onWindowResize() updates the aspect ratio of the camera and the size of the Renderer. It is called by the main each time the window is resized.

```
this.onWindowResize = function() {
  const { width, height } = canvas;
  screenDimensions.width = width;
  screenDimensions.height = height;
  camera.aspect = width / height;
  camera.updateProjectionMatrix();
  renderer.setSize(width, height);
}
```

SceneSubject: *SceneManager* is responsible exclusively for setting up and updating the scene, it doesn't know anything about its content. Every logical

component inside the scene should have its own separate component.
A SceneSubject has a basic interface:

- a constructor that takes a *Scene* object.
- a public method called *update()*.

Here are basic structure of a *SceneSubject*:

```
function SceneSubject(scene) {
  const mesh = new THREE.Mesh(geometry, material);
  scene.add(mesh);
  this.update = function() {
    // do something
  }
}
```

5 Tools, libraries and assets

In the process, I used textures, fbx file open source terrain generation library. Here are the list of external libraries and assets that used:

1. *FBXLoader.js* is used to load FBX model into the scene.
2. *ImprovedNoise.js* is utilized by *Terrain.js* to generate noise for terrain.
3. *Terrain.js* is open source random terrain generation library
4. Open source rigged spider model is used

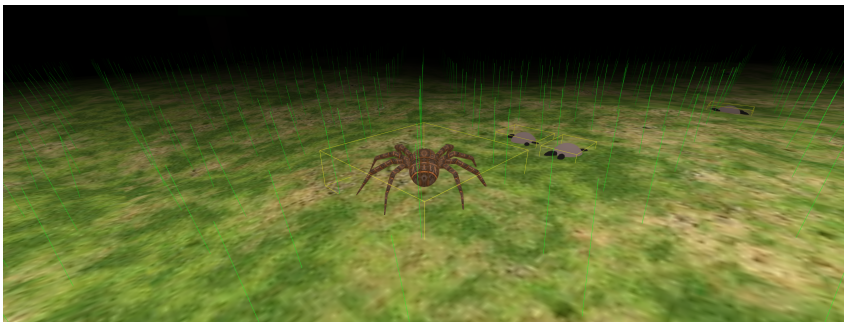
6 Technical details

6.1 Terrain hugging

Since terrain is not flat along the XZ plane, scene subjects needs to placed on to the terrain. If subject is mobile, it has to be performed inside of the *update()* function. For that purpuse, I used Three.js RayCaster class to cast a ray to the ground, which help me to determine Y position of objects.

6.2 Collision detection

I used Three.js RayCaster class to detect collision with a collidable mesh which consists from all the trees in the game. In the *update()* function, from the origin of the spider two ray being cast to the both front and back side and from the result of the RayCaster distance to the ray intersection is being checked. Furthermore, to determine the collision with the enemies and the stamina ball, bounding box intersection between those two are constantly checked for all to detect the intersection.



6.3 Animation

The animations for the project consist of the spider and insect movement. Animated spider 3D object imported via Three.js *FBXLoader()* and loaded into *AnimationClips*. Each *AnimationClips* hold the data of certain activity of the Spider. List of spider animation clips are shown in the figure 3.

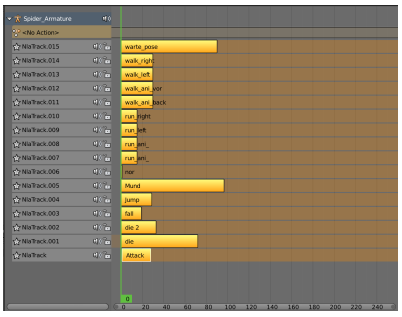


Figure 3: Animation clips

The stored data forms only the basis for the animations - actual playback is controlled by the *AnimationMixer* which can control several animation simultaneously. The *AnimationMixer* itself has only few properties and method, because it is controlled by *AnimationActions*. By configuring an *AnimationAction* you can determine when a certain *AnimationClip* shall be playing, paused or stopped on the mixer.

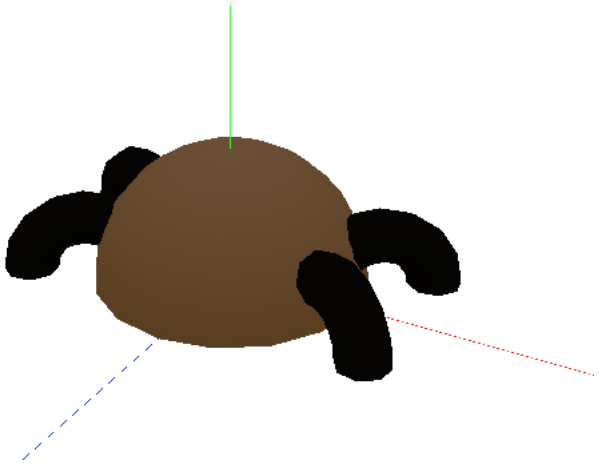


Figure 4: Enemy

One of the requirements of the project was to create animation using plain JavaScript, for that purpose, I created insect objects as *Enemy* which can be seen in the figure 4, consist of one armature and four legs. Animation is created by moving opposite leg of the insect at every frame

```
this.moveForward = function(){
        this.clock.start();
        this.rotateAboutPoint(this.frontLeft, new
            THREE.Vector3(0,0,1), new THREE.Vector3(0,1,0),
            Math.sin(this.frontLeftAngle /2) / 10 );
        this.frontLeftAngle++;
        this.rotateAboutPoint(this.frontRight, new
            THREE.Vector3(0,0,-1), new THREE.Vector3(0,1,0),
            -Math.sin(this.frontRightAngle /2) / 10);
        this.frontRightAngle++;
        this.rotateAboutPoint(this.backLeft, new
            THREE.Vector3(0,0,1), new THREE.Vector3(0,1,0),
            -Math.sin(this.backLeftAngle /2 ) /10);
        this.backLeftAngle++;
        this.rotateAboutPoint(this.backRight, new
            THREE.Vector3(0,0,-1), new THREE.Vector3(0,1,0),
            Math.sin(this.backRightAngle /2) / 10);
        this.backRightAngle++;
}
```

To prevent insect to constantly change its rotation, following code had been implemented

```
if ((Math.sign(Math.random() - 0.5) == -1) &&
    this.clock.getDelta() > 10){
   this.rotate = true;
}
else {
   this.rotate = false;
}
```

6.4 Third person camera

Third person camera achieved by taking the offset specified and apply world matrix to it. This is affected by not only spider position but also its rotation .

```
var relativeCameraOffset = new THREE.Vector3(0,200,400);
           var cameraOffset = relativeCameraOffset.applyMatrix4(
               spider.matrixWorld );
           camera.position.x = cameraOffset.x;
           camera.position.y = cameraOffset.y;
           camera.position.z = cameraOffset.z;
           camera.lookAt( spider.position );
```

7 Game win condition

To win the game, one need to hunt atleast 20 insect within 2 minutes. After the time, two different CSS overlays, seen in the figure had been shown in compliance with the result of the game

8 Further improvements

Firstly, parameters of the game are not optimal. Game level design is a whole different area. To optimize that I had to spend quite a lot of time to test the game with different parameters. Secondly, think that would have been spent time, is rotation matrix adjustment of the spider according to the terrain. Smooth transition between different normal matrix could have been implemented which also would take quite some time to accomplish. Lastly, I would analyze time complexity of the code and try to decrease overall complexity.

9 Conclusion

In the project, simple WEBGL game is developed using Three.js. Source code of the project can be found in here. All in all, it was challenging but great experience to write a game from scratch. In the process, i learn fundamental approaches of game development in WEBGL. I am confident that those knowledge will help me in my future work.

References

[1] https://github.com/srchea/Terrain-Generation

[2] https://threejs.org

[3] https://www.cgtrader.com/free-3d-models/animals/insect/spider--2