Interactive Graphics Final Project - SuperBí

Valerio Montagliani 1759418

Try it on https://sapienzainteractivegraphicscourse.github.io/final-project-valerio-montagliani/
Select the difficulty (bottom-left of the screen) then press START to play the game

Please note that with one of the last update of Firefox (linux) I started having problems with the load of the game. So, if there are problems, please test it using Chrome or Safari.

Commands:

- Left/right arrow to move Superman
- Up/down arrow to change the moon distance
- r to raise/lower the right arm
- l to raise/lower the left arm
- p to pause the game
- m to speedup the rotation speed

Project requirements

- You can use «basic» WebGL or advanced libraries, such as ThreeJS (http://threejs.org/) or Babylon (http://babylonjs.com/)
- You can use models created with a modeler or found on-line.
  YOU CANNOT IMPORT ANIMATIONS
- The project MUST include:
- Hierarchical models
    - At least one and more complex of the model used in homework2

- Lights and Textures
    - At least one light, textures of different kinds (color, normal, specular, ...)
- User interaction
    - Depends on your theme, as an example: turn on/off lights, change viewpoint, configure colors, change difficulty, ...
- Animations
    - Most objects should be animated, in particular the hierarchical models should perfom animations that exploit their structure. ANIMATIONS CANNOT BE IMPORTED, should be implemented by you in javascript (WebGL, ThreeJS or other approved library)

Project description

I decided to develop a little game about Superman. The main goal of this game is to fly around the moon avoiding the 'kryptonite' asteroids.

In this project I used `three.js`, a cross-browser JavaScript library and application programming interface (API) used to create and display animated 3D computer graphics in a web browser using WebGL.

List of all the libraries, tools and models used but not developed by me

- GLTFLoader.js, a loader for glTF resources
- Three.js
- A GLTF model of the moon (moon_scene.bin and moon_scene.gltf)
- A GLTF model of Superman (superman_scene.bin and superman_scene.gltf)

- sky_stars.jpeg image of the sky

Description of all the technical aspects of the project

Files

```
Main folder

| README.md
| index.html //main html file
└──gltf //folder containing the models
        | | moon_scene.bin
        | | moon_scene.gltf
        | | superman_scene.bin0
        | | superman_scene.bin
└──img //folder containing images
        | | sky_stars.jpeg //image used as texture
        | | background.jpg //start screen
        | | start_button.svg
        | | select_difficulty.svg
└──scripts //folder containing the javascript files
        | | game.js //script to manage the game
        | | main.js //main script
        | | three.js //file containing the Three.js library
        | | GLTFLoader.js //file containing the loader needed to load models
└──styles
        | | index.css //style file
```

main.js

The main script `main.js` will populate the `index.html` file with an initial screen where the user can select the difficulty of the game (the speed of the kryptonite) and then start the game. Once the user press start, the function `startGame()` will be executed, creating a `scene`, a `camera`, a `difficulty` value (that are passed to the constructor of the Game class) and a `renderer` (used in the animate loop).

game.js

In `game.js` is defined the class `Game` that contains all the function needed by the game.

The constructor initialize the scene using `initializeScene(scene, camera)` that will:

- Create a directional light and an ambient light
- Create a sphere and then apply a texture from an image to it (galaxy)
- Load the model of the moon and the model of superman using `modelLoad()` (uses GLTFLoader)
- Create a bounding box for the superman model (Box3)
- Create 5 obstacles using `createKryptonite()` and 5 bonuses using `createBonus()` (stored inside the Group `objectsGroup`)
- To better simulate the kryptonite gemstone and obtain a glass effect, I decided to as material:

```
KRYPTONITE_MAT = new  THREE.MeshPhysicalMaterial({
        transmission:  .5,
        thickness:  1.5,
        roughness:  0.07,
        color:  0x83f52c
})
```

- For each obstacle and for each bonus a bounding box (Box3) will be created (then stored inside the array `objectsBB`) for the collision management.

The collisions between Superman and the kryptonite/bonus is managed by `checkCollisions()` where for each element of `objectParentBB` will be performed the function intersectsBox with Superman: when this function returns true means that there is a collision. If the collision is between Superman and the kryptonite the health of superman will decrease of 10, else if the collision between Superman and a bonus, the score will be increased by 10.

The objects' moovement is managed by the function `updateObjects()` where the group's position is updated by the selected difficulty times the current time. When the objects exceed the Superman position (object z position > 0) the function reset their z and x position (calling the set functions), simulating the spawn of a new object.

The animation are performed inside the funcion `animate()`. There are several animations:

- Moon rotation
- Superman raise from the moon to the starting point
- Superman raise/lower his arms
- Superman body fluctuate
- Superman's cape fluctuate
- Shake animation when Superman hits kryptonite

Description of the implemented interactions

I implemented the following interactions:

- Left/right arrow to move Superman
  - This will modify the x position of superman
- Up/down arrow to change the moon distance
  - This will modify the scale and the y position of the moon
- r to raise/lower the right arm
  - r will trigger an animation that will lower or raise the right arm
- l to raise/lower the left arm
  - l will trigger an animation that will lower or raise the right arm
- p to pause the game
- m to speedup the rotation speed
  - m will increase the value of the x rotation