Sommario

Angry Bird – lost in the desert

Below a short description of my solution for the final project of Interactive Graphics. The unique html page accessed for the game is **index.html** . When accessed is showed a div **scene2** that contains a button "START GAME" and a summarized description of how to play the game.
Is called a the function *preInit()* that add a listener to the start button to catch user interaction.
When the "START GAME" is pressed the button and the description are no longer showed in the page; is showed the div **scene** in which will be added the elements to play and is called the function *init()* to start.
The **scene** div element has a first section containing:

- A button "EXIT" that ends the game
- A table in which there are the details of:
    - SCORE – how many points are reached during the game
    - LEVEL – in which level we are
    - LIFE – number of "life" still available

Play the Game

The game can be played following the rules below:

- The main hierarchical object in the scene is a red "angry bird", lost in the desert. You can modify the position of the bird up and down on the scene using the keyboard arrows "up" and "down". It is allowed only this kind of movement for the bird and it is controlled entirely by the user that play the game.
- You can pause the game in each moment, pressing the keyboard key "space". In this case is showed a message that the game is in pause and can be resumed pressing the keyboard key "space": this allows to restart to play.
- The Score is incremented when a "disk" is took by the bird, every "disk" gives 5 points.
- The Level is incremented every 100 points reached, every time the level go up we have an increment in the speed of the game (rotation of the sky, rotation of the desert, position of the disks and the sphere added in the scene).
- The Life is decremented every time the bird collides against a rock sphere.
- The user have 3 lifes: when they terminate, the game ends and is showed a GAME OVER message with the detail of the SCORE and LEVEL obtained.

Technical Description

Below the technical description of the logic implemented.

Init Function

The *init()* function is called when the page **angryGame.html** is loaded. Here are added the listener to the elements present in the main page, that are:

- gameOver: to show GameOver message
- totalScore: to show the score when the game ends
- actualLevel: to show the level when the game ends
- stopPauseGame: to show a message when the game is stopped
- alarmMsg: to show a message when the bird collids with a rock sphere and loses a life
- pointWin: to show a message when the bird take a disk and obtains 5 points to add to the score

The button pageReset (EXIT) ends the game resetting the number of life and calling the method *showGameOver()*.

In *init()* are also called the functions that create the objects that will be visualized and manipulated during the game, that are:

- initialize();
- createScene();
- createLights();
- createDesert();
- createSky();
- createAngryBird();
- createDisk();
- createSphere();
- loop();

The Scene

The scene is created through the function *createScene()* , in which are used different methods available in THREE js library. The scene have dimension that is adapted to the current dimension of the browser window: for this purpose is also used the method *handleWindowResize()* to handle the update of the size of the browser window (to adapt the scene to the new dimension).

Is moreover used the THREE js library method *PerspectiveCamera*, for add perspective projection (to mimic the way the human eye sees). Is also used the THREE js library method *Fog*, to make the scene more real. At the end of the function we have the render, that displays all the scene.

3D Objects

For all the 3D objects in the scene I used a mesh, which is a combination of geometry and material. I used the following function to create and add the 3D objects to the scene.

*function **createAngryBird**()*

- Object created: **AngryBird**
- Hierarchical model: yes

The AngryBird is the object that will be managed by the user to play the game. Is a hierarchical model, composed by a body, a beak, two wings and a tail. There is a principal SphereGeometry (the body) that is modified using a transformation matrix to have an oval figure. The other elements are BoxGeometry, in some cases I have modified the position of the vertices (see the wings and the beak) to give a shape more

similar to reality. To all the elements of this objects is applied a PhongMaterial shader. Creating and adding all the element inside the AngryBird class we have that all the components are treated in the scene like a unique entity.

The AngryBird object is instantiated more than one time: one to construct the principal bird (red) that can be moved by the user, a fixed number of birds (whites) are also created and added to the sky to simulate the presence of other birds in the scene that move in the opposite direction of the principal bird..

*function **createDesert**()*

- Object created: **Desert**
- Hierarchical model: no

In this function is created an object Desert and it is added to the scene: the Desert is simply a cylinder placed at the bottom of the screen. I applied a rotation matrix to this element to fix the rotation of the object around its x axis (to simulate the movement of the other objects on the screen, like the bird). Is applied also a PhongMaterial shader.

*function **createSky**()*

- Object created: **Sky**
- Hierarchical model: no

In this function is created an object Sky and it is added to the scene: the Sky is a generic 3D object in which are added other elements, a fixed number of birds that have the same structure of the angry bird but they simulate their movement in the opposite direction to the main bird.

*function **createDisks**()*

- Object created: **Disk** / **DiskOwner**
- Hierarchical model: no

In this function are created a set of disks, that are used in the game to accumulate points and to level up. We have a base class Disk and a class DiskOwner that manage the pool of disks added during the game. The DiskOwner class have two methods defined, addDisk and diskAnimation, that will be explained in the section related to animation.

*function **createSphere**()*

- Object created: **Sphere** / **SphereOwner**
- Hierarchical model: no

In this function are created a set of spheres, that are used in the game as obstacles that must be avoided by the bird. We have a base class Sphere and a class SphereOwner that manage the pool of spheres added during the game. The SphereOwner class have two methods defined, addSphere and sphereAnimation, that will be explained in the section related to animation.

Animations

The animation started when in the function *init()* is invoked the method *loop()*.

*function **loop**()*

In the loop function is managed the deltaTime that is used in the animation of the rock spheres and the disks. Only if the game is “enabled” (not in pause or ended) is continuously checked if to add new disks or rock spheres in the scene. Are used some variables that tries to put the two different kind of objects in the scene not in the same place (after several seconds it could happen, particularly when the user leave the game active and not interact with the game because the disks and spheres continue to be added to the

scene). To add new disk and sphere objects are used the methods *addDisk()* and *addSphere()* that are described in the specific paragraphs.

In this function are also called the methods *updateBird()* , *updateDistance()* , *diskAnimation()* related to the disks , *sphereAnimation()* related to the rock sphere, is updated the speed of the game and is updated the z axis rotation of the desert and the sky (they gives the illusion of the movement of the bird in the scene).

*function **addDisk**() and **addSphere**()*

These two function are about identical. For disks is generated a random number of elements that will be added in this iteration, for spheres the number is always equal to 1 (to not add too many of obstacles in the scene!). Using some mathematical functions and calculations is defined the position of the objects that are included in the scene, in order to show them between the desert surface and the up limit of the scene.

*function **updateBird**()*

This method is essentially used to update the variable birdSpeed, used in the loop function, and to modify the structure of the angryBird hierarchical model. The bird have the wings that simulate the movement up and down (inside a certain range) and has the tail that rotate right and left (respect to the body of the bird).

*function **updateDistance**()*

Here is updated the variable distance, more important because is used to define when / where insert new disks and rock sphere in the scene.

*function **diskAnimation**()*

In this function are moved the disks still in the scene to see them in the right position after that they are been added and is checked if in the current frame the bird is near enough to one of them (I setted a distance check that if is reached it allows to “took” of the disk by the bird). If a disk is reached:

- Is updated the score calling the method *updateScore()*
- Is showed in the scene the message “Score increase!” for a specific interval of time, its opacity is diminished to give the disappeared effect. Having several disk very close each other is showed only a “win” message at a time (usually when the first disk is took).
- The disk is removed from the scene

*function **sphereAnimation**()*

In this function are moved the rock sphere still in the scene to see them in the right position after that they are been added and is checked if in the current frame the bird is near enough to one of them (I setted a distance check that if is reached it simulate the collision between the bird and the sphere). If a collision occurs:

- Is updated the life calling the method *updateLife()*
- Is showed in the scene the message “Life Penalty!” for a specific interval of time, its opacity is diminished to give the disappeared effect. Is moreover updated the bird position on x axis subtracting to its current position the value -30 (to simulate the impact) and the object bird is hidden and showed for the same interval of the penalty message to simulate the blink effect.
- The sphere is removed from the scene

NB: if the LIFE value become 0 is called the method *showGameOver()* from the function *move()* , described in the section **User Interaction**.

*fuction **updateScore**()*

Here is increased the score (5 points every update) and is updated the value showed for it. Every 100 points is called the method *updateLevelAndSpeed()* .

function ***updateLife****()*

Here is decreased the number of life (the game starts with 3 lifes, every time is called this function is substracted one unit) and is updated the value showed for it.

function ***updateLevelAndSpeed****()*

Here is increased the level (every time is called this function is added one unit) and is updated the value showed for it. There is even the update of the speed of the game: increasing the rotation speed on the z axis for the desert and the sky and increasing the variable speed used in the method that animates disks and spheres we have the perception that the bird moves faster and this allows to increase the difficulty of the game.

function ***showGameOver****()*

Here is stopped the animation of the objects in the scene, is used a timer interval to simulate the fall downwards of the angryBird (rotating on itself). Is at the end showed a message that summarize the SCORE and the LEVEL reached.

User Interaction

The user can modify the position of the angryBird object up and down in the scene using the keyboard arrows “up” and “down”. It is useful to avoid rock sphere obstacles and to take the disks to accumulate points.

function ***move****()*

The movement is allowed by two variables UP and DOWN that enable to update the y position of the bird and its inclination in the scene updating the rotation over the z axis (if go up it has the beak facing upwards, if go down it has the beak facing downwards).

If LIFE value become 0 is called the method *showGameOver()*.

function ***onkeydown****()*

It manage the case in which the user is pressing a keyboard. It is possible to check the key pressed by its key, identifying the correct movement to be performed by the bird.

function ***onkeyup****()*

It manage the case in which the user stop to press a specific keyboard key. In this case the variables UP and DOWN are setted to false (based on the keyboard key no longer pressed) and the position of the bird return to be parallel to the base of the scene updating the rotation over the z axis.

function ***manageAnimation****()*

It manages the case in which the user press the keyboard key “space”. In this case the game is stopped and is showed a message “**Press Space to Resume**”: pressing the same keyboard key the game restart.

Lights, Shadows and Texture

I used HemisphereLight for the atmosphere, a DirectionalLight for the shadows (creates a shadows for the objects angryBird, disks and spheres in the desert surface) and AmbientLight. I tried with different combination of colors and values to find a good look for the entire scene. Lights and shadows are setted in the *createLightsAndShadows()* method.

As texture I defined a function *configureTextureRocks()* in which I create a texture / image from typed array Uint8Array , to have a pixeled dark effect.

The texture is used every time a new rock sphere is created, I applied a MeshPhongMaterial shader to give to this element the same aspect as the other objects in the scene (a shadow created by the light present in the scene, the rock sphere create a shadow in the desert surface).

The material of the disks are rendered with custom shaders (using the function ShaderMaterial and creating custom code in section fragmentShader and vertexShader) to have a particular effect for these elements and distinguish them from the other objects in the scene.

Environment, Libraries, License, Style

To develop the project I used the **library** Three.js :

https://threejs.org/

The **license** added to the project is *GNU General Public License 3.0*.

The game has been tested on the following **browser**: Chrome, Firefox, Internet Explorer. In this last there are some warning, but the game works.

NB: as the interaction of the user is managed through the keyboard arrows “up” and “down” is necessary to play the game with a device that has the keyboard .

For the style of the page I created a css file (angryBird.css) where are defined the aspect of the scene, of the texts showed during the game, of the table that shows the status of the game and of the buttons present.