Foxy Jumper

Interactive Graphics final project a.y. 2020/2021

Frabotta Andrea 1713373 - Scaccia Flavio 1961720

Summary:

Introduction:

For the final project we chose to remaster the famous game called 'Doodle Jump' whose goal is to reach the highest altitude possible by jumping over platforms.

Although such a game is totally developed in 2D and with very simple models, so we have decided to modify it adding a 3D model of a fox, of an enemy (rapresented by a trap) and of the platforms.

To be coherent with the original game, jumps are automatically executed any time the fox lands on a platform.

The user can move the fox horizzontally through the use of the arrows of the keyboard.

There are three types of platforms:

- the 'real' platform, that will allow the fox to execute the simple jump,
- the 'superJump' platform, that will allow the fox to perform a higher jump
- the 'vanishable' platform, that will disappear as soon as the fox jumps over them.

Platforms will be randomly generated any time the camera moves up, each new platform will be placed at a random height with respect with the previous one; such random height is computed in the range [1,simpleJump], in order to always make possible to the fox to be able to reach at least one new platform.The x coordinate of new generated steps is random in the range of the visible width of the scene.

The user loses if the fox hits a trap, in that case the fox will fall and the user cannot do anything, or if the fox reaches the bottom of the visible scene.

The visible width/height of the scene is computed by a procedure, located in utils.js, that uses the position of the camera and the field of view.

The game is characterized by an increasing level of difficulty according to the score, such difficulty is implemented through some variables that affect the density of the new produced platforms and their type; in particular, the more you go on, the more likely is to meet vanishable platforms and the less likely is to find real or superJump ones.

Menu and user interactions:

As soon as the user opens the link, a main menu is presented; here the user has three options:

- Change background choosing among three alternatives: a skyscraper, a bricks wall and a colored wood theme;
- Play the game;
- Visit the 'About' page, where he can find the tutorial and other informations like credits, details about the developers and about used libraries.

Furthermore, in the main menu, the user can activate/deactivate the audio using the icon located in the top right corner of the page.

User commands during the game are handled through the document.onKeyUp and document.onKeyDown events, using the key codes for the left arrow and the right arrow that allow the user to move the fox.

Scene set-up:

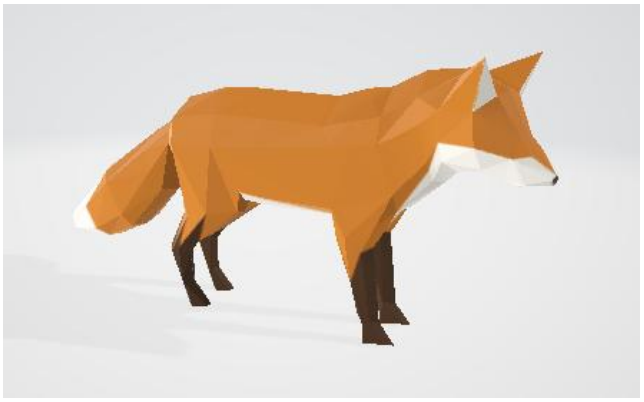
The environment of the game includes a background (a three.js plane object) on which a texture is applied and a ground, with similar implementation.
In the scene we have also applied a further background simulating a sky and atmospherical fog.

We used a perspective camera placed in front of the background plane and always looking at the center of the canvas, this is achieved by properly setting the lookAt function; its y position is updated (raised) any time that the fox overcomes the center of the canvas using a tween animation that follows the jump of the fox.

The entire scene is illuminated by an ambient light and a directional light, all these lights are implemented using Three.js library, respectively with the following functions: HemisphereLight and a DirectionalLight.

Models:

In our game, we have implemented two 3D hierarchical models downloaded from the open source web site called “Sketchfab” as a gltf file. To include such models in the project, we have imported the gltf files into the code using GLTFLoader provided by Three.js.



For both models we have retrieved all the nodes composing the structure.
In particular, for the fox, we have stored the IDs of the nodes in the fox_dic that allows us to directly access them in order to use them in the animations or for the setup of the models in the scene.

Resources:

- **Textures:** We have applied textures (taken from the open source website "Textures.com") to several objects inside our scene:
    - a grass texture for the ground;
    - three possible textures for the background (depending on user's choice);
    - three textures for the various type of platforms (real platforms, superJump and vanishable ones).

  Each texture is a combination of texureMap, normalMap and roughnessMap.

- **Sounds:** The game is suited with several sound effects taken from the website "mixkit":
    - gameOpener sound: played at the beginning of each new game;
    - Jump sounds: there are three types of sound played after the collision between the fox and the various types of platforms;
    - Trap sound: played when the fox hits the trap;
    - Fall sound: played during the fall after the collision between the fox and the trap;
    - gameOver: played when the user loses.

  The sounds are only played if the volume of the game is active, the user can activate or deactivate it in the main menu.

Animations:

The project contains some animations, all developed using the Tween.js library:

- **Fox movement:** This animation is composed by two different functions, 'left' and 'right' each containing a single tween that update the x position of the fox. Such functions are called any time the user presses an arrow of the keyboard.
- **Fox rotation:** This animation is executed by a function containing two tweens, rotateRight and rotateLeft whose goal is to let the fox turn around any time it is required to move towards a direction but it is oriented in the opposite one.
- **Fox jump animation:** This animation is the most complex one, it is executed by the jump function (triggered by a collision) and it is composed by several chained tweens:
    - Bending: This tween is responsible for the preparation of the jump, here the fox bends its *arms* and *legs* and rotates the *neck*.
    - Extending: as soon as the bending tween is over, extending and jumping tween are executed, in particular, this tween is responsible for the extension of the legs during the initial phase of the jump.
    - Jumping: as already said, this tween is executed as soon as the bending tween is completed, this is responsible for the actual jump, the y position of the fox is increased and the tail moves towards the body. In order to make the jump as realistic as possible, the easing property of this tween is set to quadratic.Out; this gives the impression of the vertical speed of the jump gradually slowing down. During the jump the tail gets closer to the body simulatinf the effect of the gravity on it.
    - GravityFalling: as soon as the jump tween is over, the fox begins its fall, the corresponding tween decreases the y position of the fox and the tail raises up giving the effect of the friction with the air. Similarly to what already done with the jump tween, in order to make it more realistic, the easing property is set to quadratic.In; in this way the fall speed increases with time. During the fall the tail is animated as well.

- **Fox fall:** This function is executed when the fox hits a trap and the game is over, the fall is similar to the gravityFall but it is faster, giving the idea of a free fall.
- **Camera movement :** This function is used to update the y position of the camera in order to follow the progression of the game.
- **Trap movement:** this is a simple tween constisting in a linear and horizontal movement of the trap, this movement is continuous so we used the yoyo property of the tween in order to let the animation repeat.
- **Trap rotation:** while translating, the trap is also roting along its axis; as for the movement, this tween has the yoyo property set.

The chaining among the various tween costituting the same animation is performed through the 'chain' method.

Collisions:

Collision detection is a crucial aspect of this project, to handle this we have used the library Physijs.

For the two models, the fox and the trap, we have created invisible physijs BoxMesh.

We have created physijs BoxMesh geometry also for the platforms, on which we have applied the already mentioned textures.

All such Box Meshes have their own collisionEventListener that allows us to detect the collision and handle them, in particular, if the fox box has detected a collision with a platform during the gravityFall the jump function for the animation will be triggered; otherwise, if a collision with the trap has been detected the fall function will be triggered and the game will end.

Note: in order to be coherent with the original game we have taken inspiration from, we have decided to 'break' some physics rules by ignoring the collision between the head of the fox and the platforms, this means that when going up the fox goes through them.

Testing:

The game has been developed only for desktop and not for mobile devices.

It has been successfully tested on Chrome and Opera browsers.

Libraries:

- Three.js;
- Physijs.js - Used to perform collisions;
- Tween.js - Used to create animations of the fox and of the trap;
- GLTFLoader.js - Used to load the hierarchical models.

Credits:

- https://threejsfundamentals.org/
- https://threejs.org/
- https://chandlerprall.github.io/Physijs/
- https://github.com/tweenjs/tween.js
- https://github.com/tweenjs/tween.js/blob/master/docs/user_guide.md
- https://sketchfab.com
- https://www.flaticon.com
- https://mixkit.com
- https://www.textures.com