JumpMan

Interactive Graphics - Final project

Simone Bartolini 1752197

1 Libraries

The game is developed in javascript using three.js for 3D graphics. OBJLoader2.js, MTLLoader.js and MtlObjBridge.js from the three.js repository are also used for loading objects and materials and OBB.js, also from the three.js repository, is used for collision detection. Stats.js is used to show the FPS. Finally, tween.js is used for interpolation of animation keyframes.
The project contains various snippets of code from the three.js documentation and from the threejsfundamentals.org website as they have been my primary source for learning how to use three.js.

2 Assets

Various royalty-free assets have been used in this project:

- The *break.wav* sound effect has been obtained from freesound.org and created by user JustInvoke;
- The *crack.wav* sound effect is a cropped version of *break.wav*;
- The *hit.wav* sound effect has been obtained from freesound.org and created by user Cigaro30;
- The *spring.flac* sound effect has been obtained from freesound.org and created by users qubodup and cfork;
- The *breakableStepTopMap.png* texture has been obtained from opengameart.org and created by user qubodup;
- The *breakableStepSideMap.png* texture is a cropped version of *breakableStepTopMap.png*;
- The *crackedStepTopMap.png* texture has been obtained by overlaying on top of *breakableStepTopMap.png* this image from seekpng.com;
- The *crackedStepSideMap.png* texture has been obtained by overlaying on top of *breakableStepSideMap.png* this image from pinclipart.com;
- The *crackedStepSideNormalMap.png* and *crackedStepTopNormalMap.png* textures have been generated from this and this image respectively using this tool for generating normal maps;

- The *cloudMap.png* texture has been designed by me starting from this image from freepik.com;
- The *columnNormalMap.png* texture is a cropped version of this image from 1004259platformevaluation.wordpress.com;
- The *groundMap.png* texture has been obtained from opengameart.org and created by user Cethiel;
- The *movingStepMap.png*, *movingStepNormalMap.png* and *movingStepSpecularMap.png* textures have been obtained from this texture pack on opengameart.org created by user JosipKladaric;
- The *realStepMap.png* texture is a cropped version of this image from flickr.com, the *realStepNormalMap.png* texture has been generated from the same image using this tool for generating normal maps.

Some of the assets have been created by myself, in particular:

- I have designed the *starMap.png* texture using Affinity Designer;
- I have designed the *hand.mtl*, *hand.obj*, *hand.png*, *head.mtl*, *head.obj*, *head.png* materials, objects and textures using MagicaVoxel and taking inspiration from this model on opengameart.org created by luckygreentiger.

3 Gameplay description

The goal of the game is to climb a spiral staircase as high as possible without falling. The player character jumps automatically and by pressing the left or right arrow key, or by tapping the left or right side of the screen on touchscreen devices, the staircase rotates. To make the game more challenging there are various types of steps:

- **Normal steps:** as the name implies these steps act as one would expect, they are static and the player can jump in them as many times as he wants;

- **Moving steps:** these steps move up and down;

- **High jump steps:** these steps will make the player jump higher;

- **Breakable steps:** the player can jump on a breakable step only one time, the next time it will break;

- **Fading steps:** these steps fade in and out periodically, the player needs to time the jump right to avoid falling;

- **Fake steps:** these steps have collision detection disabled, so the player needs to avoid them or he will fall.

4 Technical description

4.1 Hierarchical models

In the game there are two main hierarchical models, one for the player character and one for the staircase.

4.1.1 Player character

Figure 1 illustrates the hierarchical model of the player character. The head and hands models are imported .obj files as described in the assets section. The torso, the feet and the upper and lower arms and legs are just simple box meshes. The container and the various pivots are THREE.Object3D() objects and are used to define the relative positions of the various parts of the model and the rotation points used for the animations.

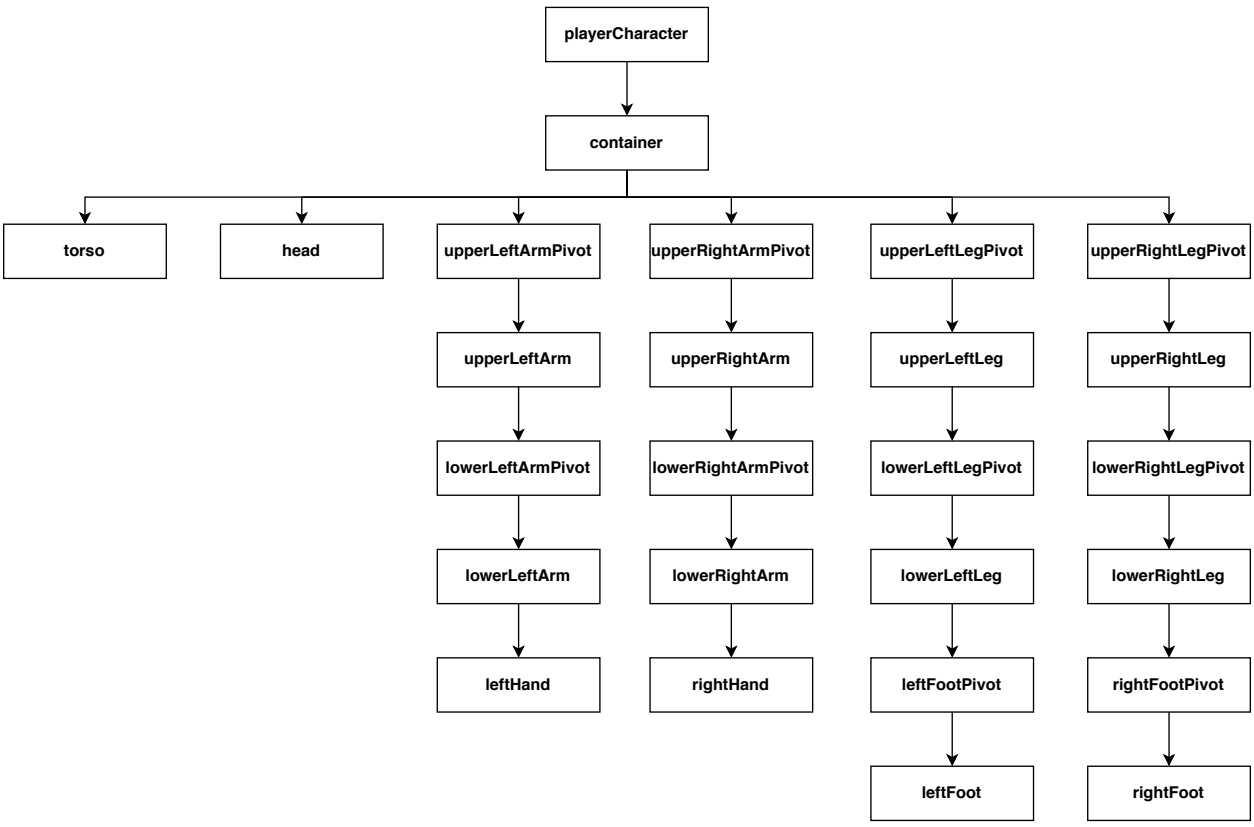


Figure 1: PlayerCharacter hierarchical model

4.1.2 Staircase

Figure 2 illustrates the hierarchical model of the staircase. It is composed of a central column, which is a simple cylindrical mesh, and a series of steps, with every step being a child of a THREE.Object3D() object called pivot that is used to define the position and rotation of the step relative to the center of the column.

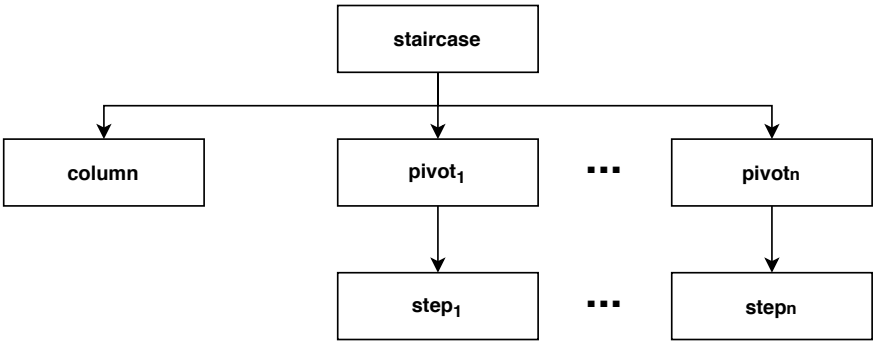


Figure 2: Staircase hierarchical model

The type of each step in the ladder is selected probabilistically by the stepTypeGenerator object. The different types have different odds of being selected and the probabilities change as the player progresses in the game to increase the challenge.
The model of each step depends on the type: normal steps, moving steps, fading steps, fake steps and breakable steps (when not broken) are simple box meshes. When a breakable step is broken it is replaced with a new model, called brokenStep, which is made of two halves that are extruded, using three.js' ExtrudeBufferGeometry class, from a flat shape having a sawtooth pattern on the cracked side (Figure 3). The high jump step instead is made of three parts, a top and a bottom box mesh and a central spring mesh, which is created using three.js' TubeBufferGeometry class (Figure 4).

(a) brokenStep hierarchical model

(b) The two halves of the brokenStep

Figure 3: brokenStep

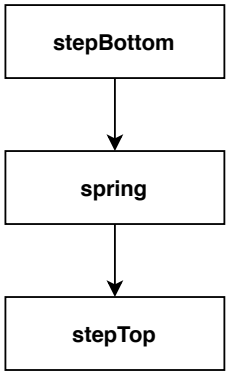


Figure 4: High jump hierarchical model

4.2 Other objects

4.2.1 Ground and Fog

The ground is just a simple plane with a flat shaded texture on top. To hide the seam between the ground and the sky a linear fog is used (Figure 5).

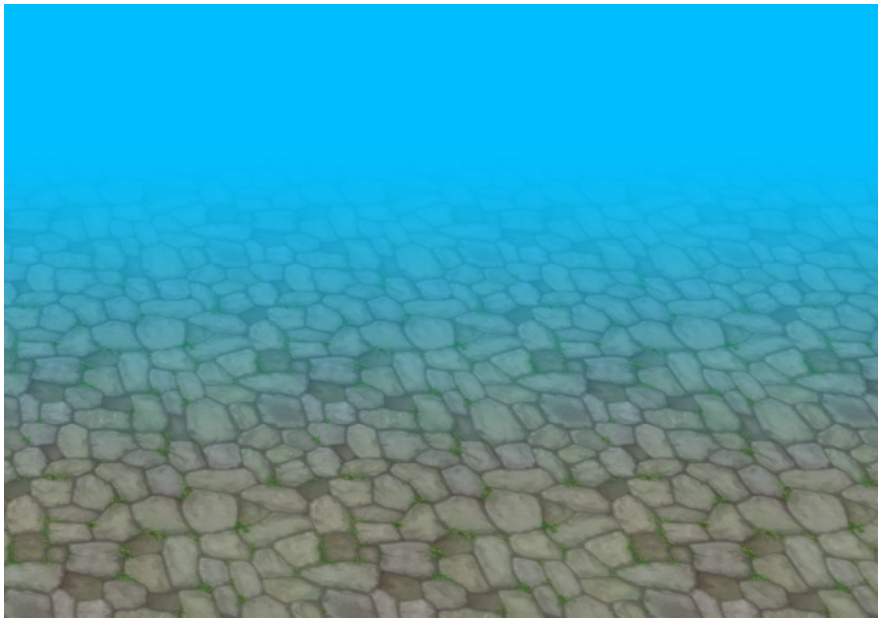


Figure 5: Ground and fog

4.2.2 Clouds

To simulate the player going higher in the sky, clouds appear once reached a height of around 200 (Figure 6). The clouds are not a 3D model but just a plane with a flat shaded texture on top, oriented to always look a the camera.

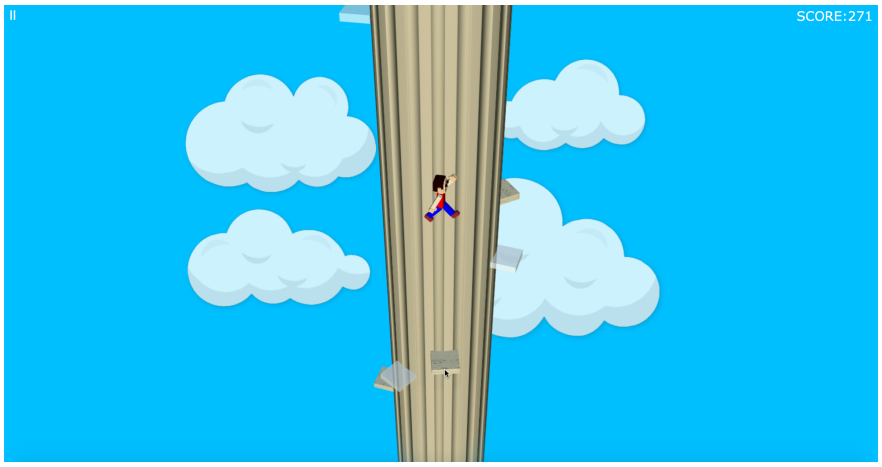


Figure 6: Clouds

4.2.3 Stars

At a height of around 700 the sky color starts getting darker and stars appear in the sky to simulate the player going into space (Figure 7). The stars are rendered using the three.js particles system.

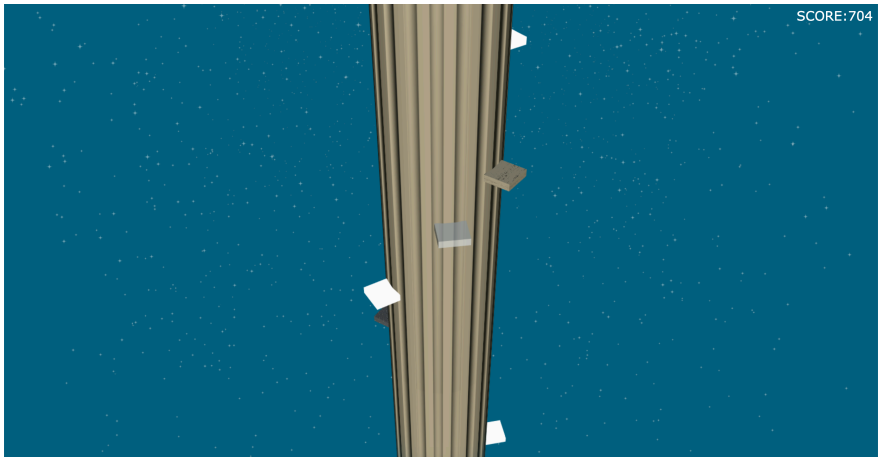


Figure 7: Stars

4.3 Lighting and textures

Every object, except the ground and the clouds, is shaded using Phong shading. There are two lights: a directional light and a hemisphere ambient light. The two colors of the hemisphere light, together with the background color, slowly change as the player climbs the ladder to simulate the player leaving the atmosphere and going into space. As for the textures, various types are used for different objects:

- the head and hands models of the player character use the color textures exported from MagicaVoxel together with the .mlt and .obj files, while the other parts of the character have just a color assigned (i.e. no texture is used);
- the column has a beige color assigned, to mimic marble, plus a normal map is used to replicate the grooves of a classical column;
- the normal steps are designed to look like if they are made out of travertine, thus the same beige color of the column is used plus a color and a normal texture to mimic the imperfections of travertine;
- the moving steps use a color, a normal and a specular map to make them look like they are made out of metal;
- the breakable steps use color textures to mimic wood. Once the player jumps on one of them for the first time, the color textures are swapped and a normal map is added to make it appear that the wood has cracked;

- as said before the ground and the clouds are rendered using a flat shaded color texture;
- a color texture is used to make the particles look like stars in the sky.

Fading, fake and high jump steps have no texture, just basic colors assigned but:

- the high jump steps have a high shininess value to make the top and bottom part look like plastic and the central spring look like polished metal;
- fake steps have an opacity value of 0.7 to make them look slightly transparent;
- fading steps are emissive.

4.4 Animations

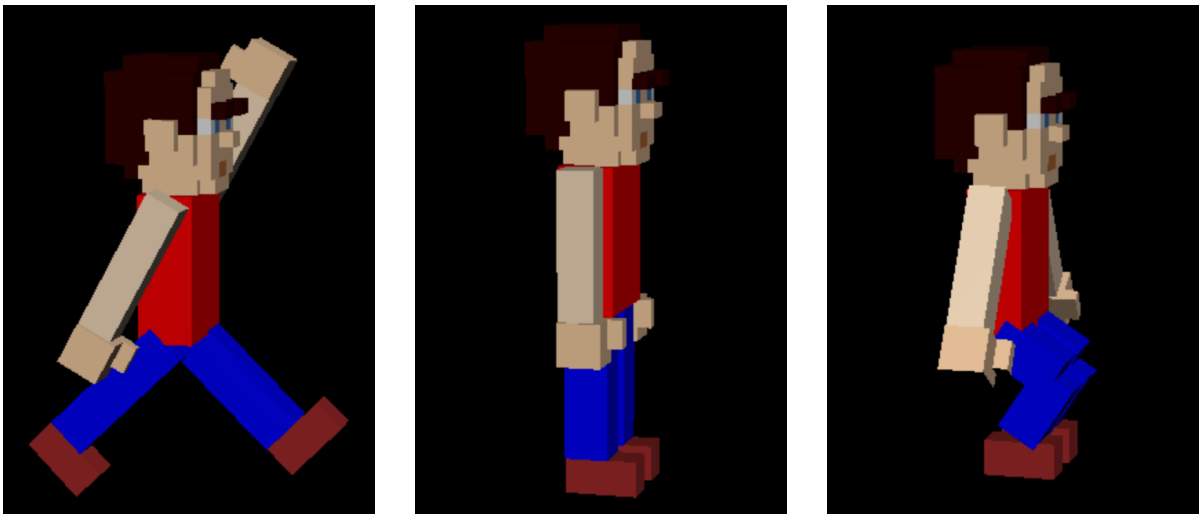
4.4.1 Staircase

As said before, the player controls the rotation of the staircase by pressing the left or right key or by tapping on the left or right side of a touchscreen device. In particular when the player press left or right the value 1 or -1 is assigned to a variable called *move*. When the player stops pressing the button (or touching the screen) the variable *move* is reinitialized back to 0. At each frame of the game the value $0.02 * move$ is added to the rotation on the y axis of the staircase.
When a collision is detected between the player character and a step that is higher than the previous one the camera is moved up. The animation of the camera movement is eased using tween.js and the quadratic ease-out function. Since the player could in theory continue climbing the ladder without falling for an infinite amount of time we will need an infinitely long staircase. To solve this problem the central column of the staircase is designed to be a little bit longer than the screen height and is moved up together with the camera to fake it being infinite (except for the first part of the game when the ground is still visible, moving the column at that time would break the illusion). Also every time a step goes out of frame it is removed while new steps are added on top of the staircase.
Tween.js is also used to animate the moving, fading, breakable, and high jump steps in particular:

- the moving steps move up and down repeatedly by 5 points with a quadratic in-out easing function;
- the opacity of fading steps goes repeatedly from 1 to 0 and back to 1 with an exponential in-out easing function;
- when a breakable step is broken the two halves of the broken step are animated to fall down and move away from each other;
- when the player jumps on a high jump step the spring is compressed, then extended and then it goes back to the initial state.

4.4.2 Player character

The animation of the player character is divided into two cycles, one defining the up and down movement of the jump and the other defining the pose during the jump. The game starts with the player character already in the air and falling down to the ground. At the beginning it falls by 15 points using a quadratic ease-in function (simulating gravity), then it continues falling at a constant speed (simulating having reached terminal velocity). Once a collision with a step is detected, the falling animation is stopped and the jumping animation starts, i.e the character goes up by 15 points (or by 44 points if it stepped on a high jump step) using a quadratic ease-out function. Once reached the end of the jumping animation, the falling animation will be restarted. In the meantime the character changes pose, interpolating between three keyframes (Figure 8). The character starts on keyframe 1 (Figure 8a), then while falling switches to keyframe 2 (Figure 8b) and then goes to keyframe 3 (Figure 8c) when colliding with a step. After that it goes back to keyframe 2 (Figure 8b) and then to keyframe 1 (Figure 8a) once reached the apex of the jump.



(a) Keyframe 1 (b) Keyframe 2 (c) Keyframe 3

Figure 8: Keyframes of the jumping animation

5 Other notable stuff

5.1 Preloading

To avoid stuttering every time a new step type appears in the game the steps materials and geometries are preloaded in the first frame of the game, by adding and then immediately removing the steps and positioning them out of the view frustum to ensure they are not visible, so that the relative shaders are compiled and the data sent to the GPU.

5.2 Tutorial

The tutorial page explains the rules of the games and shows the various steps types. This page is heavily inspired by this guide on threejsfundamentals.org on how to have multiple scenes in an html page, aligned with the text of the page. In this case the technique is used to draw the various step types near their relative description.

5.3 Leaderboard

The game has also a simple local leaderboard. At the end of a game the player can save the score, which will be added in the browser cookies. When the player access the *Leaderboard* page the scores will be retrieved from the cookies.