# SONIC RUN

Interactive Graphics - Final Project 2022

Lorenzo Bianchi 1756496

## Contents

# 1 Introduction

Sonic Run is an endless run game inspired by the famous *Sonic* franchise, created by *SEGA*. The game has been developed for the final project of *Interactive Graphics 2022* course at *Sapienza University of Rome*. In Sonic Run, the player takes role of Sonic to challenge him/herself by running through many different obstacles and enemies. Get as many rings as possible!

# 2 Environment

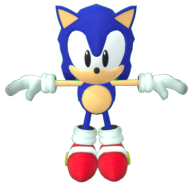
The game was developed using mainly *Three.js* library, which handles most of the models, scene and sounds. In order to load glTF models into the ThreeJS scene I used *GLTFLoader.js* and an outlines effect was added from *OutineEffect.js*[1] for aesthetic reasons. Moreover, *Tween.js* was used for smooth animations of characters and objects. Finally, a basic use of HTML and CSS was needed to implement the main menu and for aesthetic improvement.

# 3 Technical aspects

## 3.1 Models

All the 3D models are imported from *Sketchfab*[2] website, however all of them have no built-in animation nor are any animations imported. All animations are implemented by me in ThreeJS/TweenJS. What follow are all the 3D models present in the game and some basic considerations about their role in the game:

- `Sonic`: Playable Character with a hierarchical structure. The main animations he can perform are: run, jump, slide, take damage, move left and move right (3.2.1).

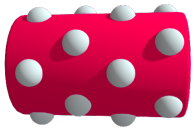


[1] *https://github.com/mrdoob/three.js/blob/dev/examples/jsm/effects/OutlineEffect.js*
[2] *https://sketchfab.com/3d-models*

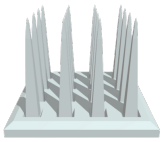
- `Eggman`: Non Playable Character with a hierarchical structure. He spawn during the run with a low probability and should be consider Sonic's main enemy. The main animations he can perform are: "grab" movement, jump, move left and move right (3.2.2).

- `Ring`: Item to collect. It spawn in groups of 2 up to 4 with certain probability. No hierarchical structure, the animation is a simple rotation around the vertical axis.

- `Cylinder`: Trap to be avoided by sliding. It can spawn in each of the three lanes. The position and the quantity to be spawned are random. No hierarchical structure, the animation is a simple rotation around the horizontal axis.



- `Spike trap`: Trap to be avoided by jumping. It can spawn in each of the three lanes. The position and the quantity to be spawned are random. No hierarchical structure nor animation.



- `Sonic's head`: Bonus item that gives Sonic an extra life. It can spawn in each of the three lanes with a low probability. If Sonic has 5 lives this item cannot be spawned. No hierarchical structure, the animation is a simple rotation around the vertical axis.

- `Tree`: Background object for countryside theme. No animation.
- `Lamp`: Background object for city theme. No animation.

## 3.2 Animations

All animation are implemented using *TweenJS* with different easing functions to provide smoother movements.

### 3.2.1 Sonic

Sonic's hierarchical model is composed by serveral nodes each corresponding to a specific bone of his skeleton. A dictionary was implemented to map some of the main nodes to the correspondent bones involved in the animations. The extracted bones involved in Sonic's animations are: spine, head, thighs, knees, upper arms and forearms. To be able to implement the different animations, such bones need to be rotated along one of the three main axis of specific angles. For this purpose, *ThreeJS* is helpful to choose the right rotation speed and the easing function for the interpolation. In order to obtain a continuous movement, as for the case of running, bones

are rotated back and forth in loop and in a synchronized way.

In general, it is always important to make sure two animations do not conflict one with the other, for this reason it is necessary to manage their execution based on the user activity. When it is needed to switch from one animation to another, e.g from run to jump, all the tweens involved in the current animation are paused, to let others perform their tasks. They will be later resumed once the other one is completed.

### 3.2.2 Eggman

As for the Sonic case, Eggman is composed by several nodes, and similarly to previous discussion, they will be used to perform Eggman's animations. The "grab" movement is given by the rotation of its upperarms back and forth of three different angles in each of the three main axis. While the jump is performed with two small rotations of the knees along the z axis together with a whole mesh position movement along x and y axis. Eggman moves back and forth from one lane to the others to add a dynamic component in the obstacles that Sonic has to face.

## 3.3 Objects generation

During the run, group of objects or Eggman's character are added to the scene with a fixed period of time. Their spawn positions along the z axis depends on Sonic's current z position. Eggman has a low probability to spawn and when it happens no other object can spawn next to him. On the contrary objects as rings, cylinders, spike traps or Sonic's head have an high probability to spawn and are the most common items Sonic will face. They can spawn in groups of at most 6 slots in a fixed arrangement. In order to assign and keep track of objects' position, each group is mapped to a 2x3 binary matrix, where the rows represent the y axis and the columns the x axis of object's position. If an entry of the matrix is set to 1, it means an object will spawn at the given position. The constraints for a group to be generated in the scene are:

1. The matrix corresponding to a group must have at least two entries set to 1, one for the rings and one for an obstacle, i.e. spike trap or cylinder;

2. Cylinders are always mapped to row 0 of the matrix while spike traps to row 1, they should have in fact different values of y;

3. There cannot be a spike trap and a cylinder assigned to the same column, i.e. with the same x-coordinate.

Finally, Sonic's head has a low probability to spawn and if it does it takes the place of rings in the group.

### 3.4 Memory management

Memory management is one of the main issue to be consider when it comes to video games. Particularly when dealing with an endless run game we should carefully achieve an efficient management of the models we are using, for a more fluid game experience. In Sonic Run potentially infinite many meshes could be loaded throughout each run. For this reason every object or chunk of terrain that is generated and added to the scene is stored in a group[3] or an array for a limited amount of time. The object is finally removed from the scene and from the group once it is no longer visible from the camera.

### 3.5 Collisions

In Sonic Run collisions between meshes are managed using "collision boxes". To each character and to each item is associated a collision box at the moment they are added to the scene. A collision box is a simple transparent cubic mesh that resemble the dimension of the mesh it refers to and it is never visible in the scene.


Figure 1: Game scene with Collision Boxes

A raycaster is used to detect collisions between boxes of the various game items and Sonic. It ray-cast from a given origin, which is located at the center of the player collision box (Sonic's box), and directed towards the

[3] *THREE.Group()*

normalized negative z-axis, that is the direction in which Sonic is faced. Collisions are detected using Raycaster's class method *intersectObject()*, that checks if a given collision box has intersected with a ray. This operation is performed repeatedly for all the active boxes in the scene. If a collision with the box of an obstacle (or Eggman) is detected, Sonic loses a life and the damage animation (mesh blinking) together with damage sound effect are played. If the collision is with the box of a ring (or an extra life) the GUI is updated and the correspondent sound is played.

### 3.6 Lights and Textures

In the game are present two different lights, an ambient light and a directional light. The directional light is positioned just behind sonic w.r.t. the z-axis and with a positive value of y. It targets Sonic during the run by moving with him at the same speed along the z-axis. Depending on the theme selected by the user, the light have different intensity to emulate night time and day time. All the objects and characters cast shadows while the terrain receives it.

A fog with light blue color is located in the background to hide the horizon given by the terrain's planes and hide the spawn of objects.

Textures are mapped to the plane meshes used for the main road and the terrain. Depending on the theme selected, the game shows: city road, sidewalk, sand or grass textures.

### 3.7 Sounds

Sounds are played in response to different events occurring during the game, such as damage taken, eggman spawn, game over, etc. All the in-game sounds are downloaded from *101soundboards.com*[4] website. The theme song in background is *Green Hill Zone*, from *soundfxcenter.com*[5], and it is played in loop during the whole game.

[4]https://www.101soundboards.com/boards/10990-sonic-the-hedgehog-sounds
[5]http://soundfxcenter.com/download-sound/sonic-green-hill-zone-theme-song/

# 4 Interactions

## 4.1 Difficulty

The difficulty of the game can be set by the user in the main menu, by default is set to normal. He/She can choose between easy, normal or hard mode through the according buttons. On user click of such buttons, the global variable referring to the game velocity updates to a different value.

## 4.2 Theme

The user can choose between two main themes in the main menu: Countryside or City. In Countryside the texture loaded are sand for the main road and green grass for the terrain at the left and the right of the road. Moreover the ambient light and the directional light have high intensity to emulate sun light condition. On the contrary, in city theme, the lights have low intensity to give the feeling of night time. Textures of this themes are road and sidewalks.

## 4.3 In-game

During the game the user can use the typical WASD or UP/ LEFT/ RIGHT/ DOWN commands configuration to move, jump or slide Sonic. Moreover at any time in the game he/she can decide to pause the game using ESC command, when it occurs every sound, music and tween that is managing an animation is paused. Once ESC is pressed again everything is resumed and the game continue from to the previous state. Finally, when the game ends the user can return to the main menu by pressing ENTER.