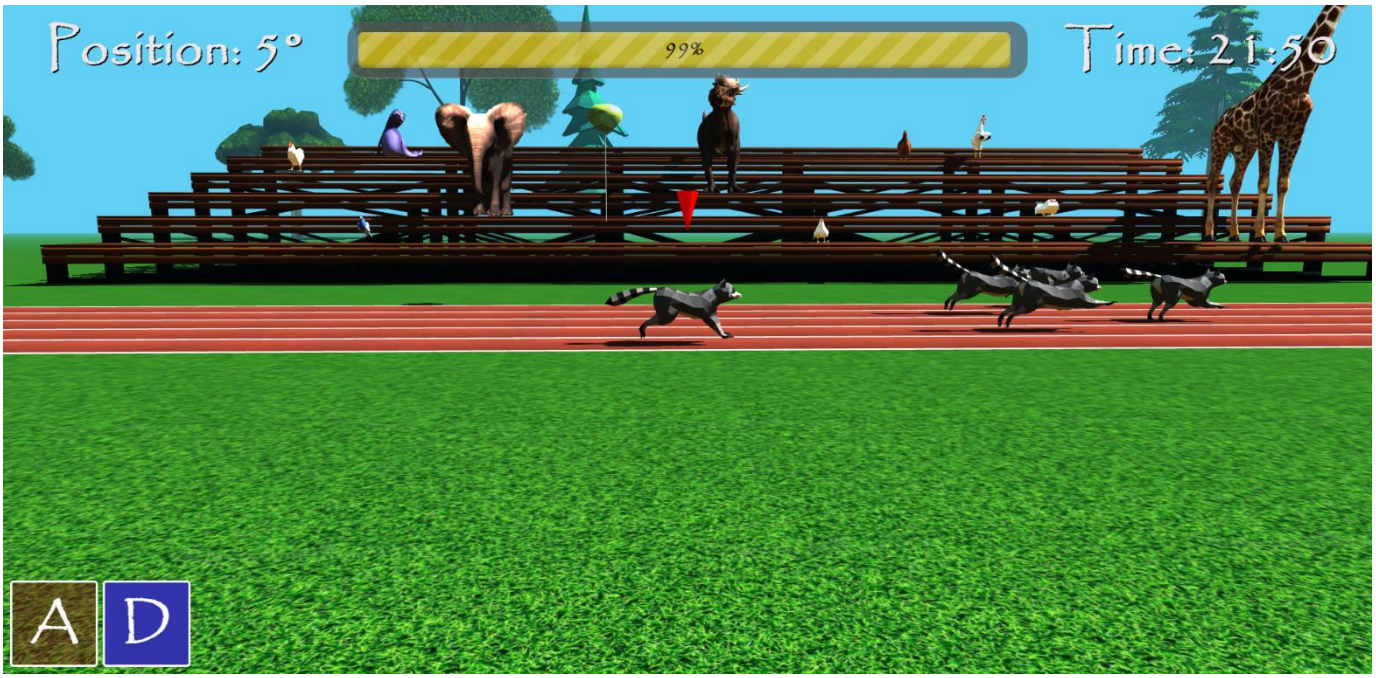
RACCOONS RUSH

Interactive graphics project



Students:

Omar Bayoumi 1747042

Cristiano Bellucci 1760390

Table of contents

1. Introduction 1
2. Technologies 1
   2.1. Three.js 1
   2.2. Tween.js 1
3. The game 1
   3.1. Camera and Controls 1
   3.2. Light 2
   3.3. Track field and Grass 3
   3.4. Animals 3
   3.5. Raccoons 4
   3.6. Trees, Finish line, Tribune, Balloons, Fog 5
   3.7. Cursor 5
   3.8. Html 5
      3.8.1. Index.html 5
      3.8.2. Main Menu 6
      3.8.3. Loading Screen 6
      3.8.4. End game screen and countdown 6
      3.9. Audios 6
4. Global Aspects 7
   4.1. Technical details and problems encountered 7
5. Levels 8
6. Conclusion 8

1. Introduction

The project was implemented on a Windows system using Chrome via the three.js library. The goal was to make a **3D game**. In the game, we will play the role of **Steve** a raccoon runner. The user will have the ability to move Steve through the **keys A and D** also **balloons** will appear on the screen, if we click them, they will burst and provide a speed boost to Steve. In the main scene, there are various **trees**, a **track** and a model for the **finish line** has been created. There is also a **tribune** on which there are **various animals**, each with a different model and a different animation, that will cheer for Steve and the other raccoons in the race. The aim of the game is very simple, we should win the race.

2. Technologies

2.1. Three.js

Three.js is a JavaScript library and an API (Application Programming Interface) that is used to create and display animated 3D computer graphics in a web browser without relying on proprietary browser plugin: this is possible due to the advent of **WebGL**. Three.js runs in all browsers supported by WebGL 1.0. Some of the main features of this API are:

- **Scenes**: add and remove objects at run-time;
- **Cameras**: perspective, orthographic, cube, array and stereo;
- **Geometry**: planes, cubes, spheres, torus;
- **Objects**: meshes, particles and more;
- **Materials**: basic, Lambert, Phong, smooth shading, texture and more;
- **Lights**: ambient, directional, point, spot and shadows;
- **Data loaders**: binary, image, JSON and scene.

2.2. Tween.js

Tween.js is a library that helps the user in the implementation of **animations**. A tween allows changing the values of the properties of an object in a smooth way. to do this just pass in input the properties that we want to change. For example, if we want to animate the rotation of a joint just choose the target value of the animation of the rotation. In addition, we should also choose the **duration** of that animation and the tweening engine will take care of finding the intermediate values from the actual to the ending point.

3. The game

3.1. Camera and Controls

The camera used is a **perspective camera**. The camera is a very important component within the game in fact it has also been "**animated**". As soon as the game starts the camera shows all the raccoons runners from right to left and then moves to the initial position chosen for the beginning of the game, always looking at the raccoon controlled by the user (Steve). Throughout the run, the camera will always be **looking at Steve**. The x-coordinate is equal to Steve's x-coordinate multiplied by 1.2. With this calculation the camera will move, initially looking at Steve from **behind** ($\text{Steve.x} = -70, \text{camera.x} = -84$), in the middle of the race it will be

Students:
Omar Bayoumi 1747042
Cristiano Bellucci 1760390

perfectly **perpendicular** to Steve ($\text{Steve.x} = 0, \text{camera.x} = 0$) and at the end of the race, it will look at Steve from the **front** ($\text{Steve.x} = 70, \text{camera.x} = 84$). In this way, we can better admire the **3D environment**.

The user will have the possibility to interact through the **A and D keys** and the **mouse click**. In the scene an **HUD** has been specially created containing: a **bar** that represents Steve's speed, the faster the user will be in pressing the above keys alternately, the faster the bar will charge. Furthermore, depending on the difficulty chosen, it will be more difficult to reach the maximum speed. There is also a system that punishes the user if he presses the same key consecutively, in which case the accumulated speed will decrease a lot.
To make the game more interactive, it was decided to introduce **five balloons** into the scene that will appear under a certain condition. The **i-th balloon** will appear if Steve reach **i ∗ 20 units** of the x coordinates of the track field. The balloons will be randomly generated within a **small area** on the opposite side of the camera position. The balloons have two animations created with **tween.js**: the first one **squeezes and stretches** the balloon while the second one **brings it upwards**. These balloons can be clicked by the user with the mouse. If the user is skilled enough to click them before they disappear from the view, he will receive a **speed boost**.
The HUD also shows the **time elapsed** since the start of the race and Steve's **current position**.
In addition, at the bottom left of the screen, there are the **A and D keys**, respectively inside a red and a blue box. If the user presses one of the keys, the frame becomes slightly transparent while the frame of the other key is highlighted. In this way, the user understands which of the two keys A and D must be pressed to avoid the penalty of the same key repeated several times. The HUD has been realized entirely in **CSS** and through the JavaScript file, its elements are updated in **real-time**.

3.2. Light

For the project have been used:

- **Hemisphere light**. It takes a **sky color** and a **ground color** and just multiplies the material's color between those 2 colors: the sky color if the surface of the object is pointing up and the ground color if the surface of the object is pointing down;
- **Directional light**. It is a light that gets emitted in a specific direction. This light will behave as though it is **infinitely far away**, and the rays produced from it are **all parallel**;
- **Point light** A light that gets emitted from a single point in **all directions**.

In different difficulties different lights were applied:

- **Easy**. The following difficulty is set **during the day**. The lights are:
  - **Hemisphere light**:
    - **Sky color, light blue**;
    - **Ground color**, **brownish orange**;
  - **Directional light:** color **white**;
- **Medium**. The following difficulty is set **during the sunset**. The lights are:
  - **Hemisphere light:**
    - **Sky color**, **sunset orange**;
    - **Ground color**, **brownish orange**;
  - **Directional light:** color **sunset orange;**
- **Hard**. The following difficulty is set **during the night**. The lights are:
  - **Hemisphere light:** the colors are the same as in easy mode, but the **intensity is 1/5**;
  - Unlike the other modes, there is **no directional light**;
  - There are **seven point lights** positioned in correspondence of the **same number of flashlights**. The color of the light is **orange fire**.

3.3. Track field and Grass

For the grass and track field were used:

- **THREE.PlaneGeometry**;
- **THREE.MeshBasicMaterial**;
- **THREE.MeshLambertMaterial**;
- **THREE.MeshPhongMaterial**;

Special **textures** have been chosen for both track field and grass. To facilitate the positioning and to correctly apply the textures to the geometries, a function myPlane() was created, which takes in input:

- **SizeX**: width of the plane;
- **SizeY**: height of the plane;
- **Texture**: path of the textures to be applied to the plane;
- **X**: x-position of the plane in the scene;
- **Y**: y-position of the plane in the scene;
- **Z**: z-position of the plane in the scene;
- **Line**: is used to correctly calculate the value to assign to the texture. This was done because the white lines in the track field have a different **thickness** than the lanes, so it was necessary to modify the **y coordinate** of the texture that was wrapped with **clampToEdgeWrapping**. This technique allows us to **stretch** a texture beyond the end of the plane to show on the lines a texture with dimensions equal to that of the lane. To correctly determine the size of the texture to be applied, a calculation was performed:
  - **Lane**: $\frac{1}{1} = 1$. In this way, the displayed texture will be normal **without the wrap**;
  - **Line**: $\frac{1}{\left(\frac{\text{lane thickness}}{\text{line thickness}}\right)}$. In this way, it will be wrapped in **proportion** to the lane and line thicknesses;
- **Repeat:** represents the number of times the texture is repeated;
- **EndLine**: is a boolean that tells us if we are working with the white vertical line at the beginning and the end of the track. **The base value is false**;
- **Grass**: is a Boolean that lets us know if we are working with the **grass** (in which case the value will be true) or with the **track field**. **The base value is false**.

To **optimize**, the **MeshBasicMaterial** is always used if shadows are disabled. This choice was made because shadows are not cast on this type of material, and it is much lighter from a computational point of view. The **MeshLambertianMaterial** is used in day mode [3.2] because it allows casting shadows and is lighter than the **MeshPhongMaterial**. In the night mode [3.2] it was necessary to use the **MeshPhongMaterial** since it still allows to cast shadows and in addition, it allows to show the light effects given by the point lights.

3.4. Animals

Other animals were also introduced in the scene:

- Five different types of **chicken**;
- A **seal**;
- A **giraffe**;
- An **elephant**;
- A **pigeon**;
- A **dinosaur**.

All the animals were animated with **Tween.js** and each animal has a **different animation**. The models are glTF models that were loaded through a **glTF_loader**. Were also loaded through an **audio_loader**, the audio that represents the **verse** of the animal. Each animal has an associated audio in this way the cheer of the animals

on the stand is more realistic. Since the project was designed to **run on Chrome**, to try to circumvent the **browser's policies**[1] on audio the audio is played after the user clicks the **play button** on the loading screen. In this way, thanks to this interaction, it is possible to start the audio within the game.

3.5. Raccoons

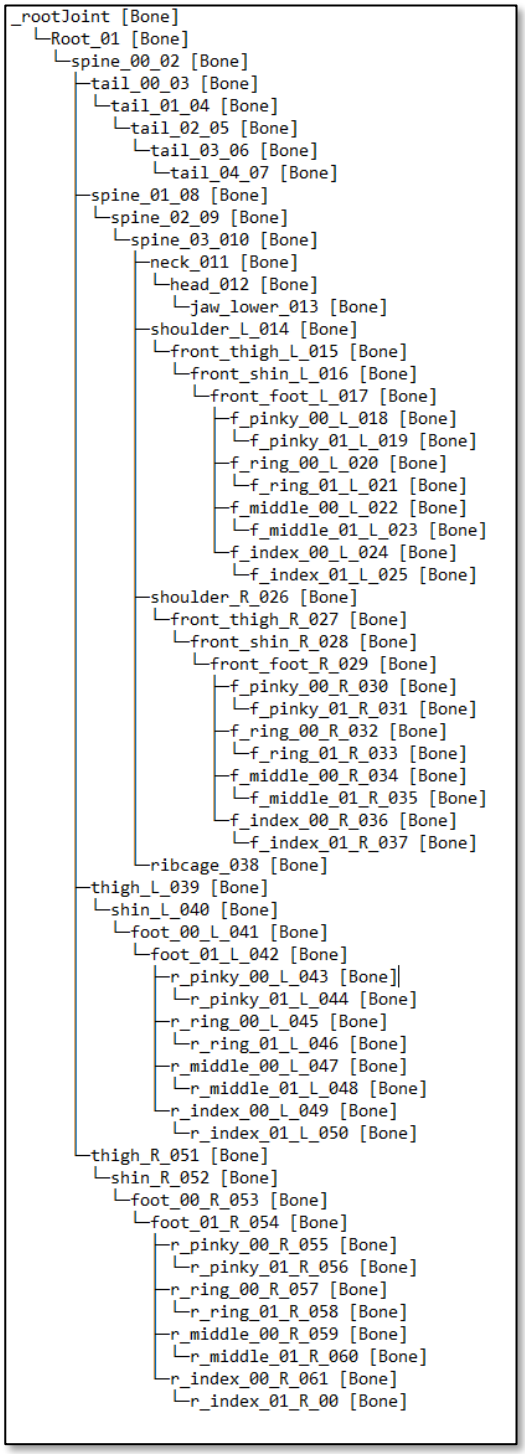


Figure 1: hierarchical structur of the raccoon.

The same glTF model was used for **raccoons**. The models are placed in the scene at the beginning of the track field, one per lane. The **closest one is Steve** [3.7]. Each raccoon has a hierarchical structure [Figure 1].

Exploiting this structure, the **joints** of the raccoon have been moved and/or rotated to make it stand **upright** and obtain its **initial position**. The **animations were do**ne by creating a distance given from the **starting point** to the **ending point**. This distance was then divided by the **number of frames** needed to complete it thus obtaining the **step of moving** to be added. Some of these animations have been structured in **phases**. You start from an **initial state** and reach other states depending on the condition. Each state has a different animation. The sequence of all states performs an animation that uses the **same number of frames as non-phase animations**, this is because the individual phases use only a percentage of these frames, and the **sum of all percentages is 100**. To give the effect of a **real run**, the joints are moved **back and forth** thanks to the use of a multiplier that assumes values **between 1 and -1** by changing the sign of the step of the movement. The change of the value 1 and -1 occurs according to the rotation of the x-axis of the left shoulder (***shoulder_L_014***).
The raccoon **runs from left to right**. Each raccoon's goal is to run **3 units** in an entire animation. So the distance traveled is calculated as follows:

$$\frac{3}{\text{number of frames needed to complete an entire animation}}$$

Depending on the difficulty the raccoons will need a different number of frames. Steve, unlike the other raccoons, will need about **twice** as many frames to complete the animation, because he has the **maximum bar speed as a multiplier**, which is **2**. The following formulas indicate how the number of frames needed changes over time to simulate a real race:

- **Steve**. The formula:
  - $frame * (barSpeed + baseSpeed + boostBalloon)$
  - **barSpeed**: current value of bar speed. The maximum value is 1.75;
  - **baseSpeed**: the base movement value, which is 0.25;
  - **boostBalloon**: cumulative value given by exploding balloons;

[1] *Chrome does not allow audio to start on a page unless there has been an interaction from the user.*

- **Other raccoons**. The formulas:
  - $difference = (speedCenter - frame)$;
  - $percentage = \frac{(100 * difference)}{range}$;
  - $frame += (Math.random - 0.5) + \frac{\left(\frac{percentage}{100}\right)}{2}$;
    - **speedCenter**: number around which the frames should be;
    - **difference**: difference between actual frames and **speedCenter**;
    - **range**: maximum value added to or subtracted from **speedCenter**.
    - **percentage**: --> value **between -100 and +100**. When the **difference** is equal to **-range** the value will be **-100**, **+100** when difference is equal to **+range**. Intermediate values are interpolated;
    - **Result**: on a random value ranging from **-0.5 to +0.5** is added or subtracted **at most 0.5** obtained by dividing the **percentage** obtained previously **by 200**. If the **percentage** is **-100**, we get as **result** a value **ranging from -1 to 0** if the percentage is **+100** the value will **range from 0 to +1**. To make the race between the other raccoons competitive and unpredictable, this random value is added to the number of frames.

3.6. Trees, Finish line, Tribune, Balloons, Fog

Inside the scene, there are also **trees**, which contribute to the setting, a **tribune** on which are placed the **animals** that cheer, and a **finish line** that was created specifically through the **Maya program** also the texture applied to the model was done using **Photoshop**. The **trees**, the **tribune**, the **finish line**, and **balloons** are all **glTF models**. Different models have been used for the trees to give diversity to the environment.
Inside the environment a **fog** has also been inserted, to avoid that the sides of the plane, representing the terrain, were visible making it look like it does not end. To do this, a fog() function was created. This is done through **Three.fog**. This class contains the parameters that define linear fog. After that, a light blue, sunset orange, or night blue color was set as the background value, depending on the difficulty chosen [3.2], to simulate the sky.

3.7. Cursor

Since all raccoons have the same model, to identify Steve, a **cursor** was used. This cursor will follow Steve throughout the race and was created as a **cone** with **three sides** to simulate a **pyramid** through a **THREE.js primitive**. This pyramid is **rotated backward** and has been animated with an animation that goes **up and down** and **rotates slightly around its z-axis**.

3.8. Html

Html is a formatting language that describes how the textual and non-textual content of a web page is laid out or displayed graphically (layout) using formatting tags. We used Html for the **start page**, the **loading screen**, the **countdown**, the **end game screen**, and the **main menu** page that will be used to choose the difficulty and enter the game.

3.8.1. Index.html

This page is opened by default from **Github pages**. Inside it a button that allows you to reach the main menu. A note advising the user to enable **hardware acceleration** has also been added. This page also serves the purpose of **allowing the background music** in the main menu to start[1].

3.8.2. Main Menu

In the **main menu**, there is a **description** of the game, **instructions**, and the choice of **difficulties**. Also, for those who have a high-performance computer, the possibility of **enabling shadows** has been added.

3.8.3. Loading Screen

During the **loading** of the various game elements, a **loading screen** is shown to the user. At the end of the loading process, the loading screen is replaced by a button[1] that, if clicked, will start the game. Since loading may take **several seconds**, the user is warned of this through a message. There are three loading pages, graphically equal, that allow starting the different difficulties.

3.8.4. End game screen and countdown

To give the user, the possibility to prepare for the race, a **countdown** has been inserted with appropriate audio. The countdown is done by updating from time to time a "div" inserted inside the HUD and positioned in the center of the screen. It has been made also animation for the countdown in fact that allows **enlarging the text**. Every time a second passes the value changes from "*3*", then to "*2*", then to "*1*". At the moment of the starting, the value changes from "*1*" to "*GO!*". For the "*GO!*" text a **fading effect** has been done. Then the value inside this "div" is deleted to avoid the appearance of text during the game.
As soon as any raccoon wins the race all animations are stopped, the sounds of the animals cheering are stopped, as well as the music that accompanies the race, the camera is moved in such a way as to simulate a **photo finish** and a new music starts. To make the whole thing more pleasant, a **semi-transparent end-game screen** has been inserted with summary information such as:

- **Time**. Time taken to reach the end. If Steve did not finish first "N/A" will be shown;
- **Position**. Steve's final position;
- **Balloons popped**. Number of balloons popped;
- **Main menu button**. Button that will allow the user to reach the main menu[1].

3.9. Audios

Various **audio files** were used to make the game:

- **Animals**. Each animal has associated audio with its own verse;
- **Raccoons**. For the raccoons, the audio taken represents the noise his paws make when they touch the track field;
- **Background music**. Three different background music were used:
  - One for the main menu;
  - One that will last from the start to the end of the race;
  - One that will be played at the end of the race during the end screen;
- **Countdown**;
- **Balloons**. Three different audios representing the explosion of the balloon.

All the audios, except for the explosion of the balloon, are loaded at the beginning of the game, thanks to the **loadingManager**, together with all the other elements and then they are started. This was done because during the design we noticed that if the audios were loaded and started on the moment, they could give **delay problems**. All audios are stored as a pair **<Three.Audio, id audio>** in the "sounds" array. We have also created a function play() that facilitates everything. This function takes in input:

- **Audio**: path of the audio file we want to load;
- **Volume**: audio volume value;

- **Loop**: if true the audio will loop, otherwise not;
- **Preload**: if true, the audio will be loaded and started on the moment, otherwise it will just be loaded;
- **IdValue**: string id given to a certain sound;
- **Delay**: value in milliseconds that represents the waiting time before the sound starts. It was inserted to make the audios of the cheering animals asynchronous.

Next, it was necessary to play the audios that were stored in the sounds array and that were not played immediately. To do this, a findSound() function was created:

- ❖ **Input**: string id;
- ❖ **Output**: audio in "sounds" that corresponds to the id passed in input.

On the value obtained, then, the command play() is used to **start** the audio and stop() to **stop** it. As for the audio elements having the **empty string as id**, since they represent only the audio of the animals on the tribune, a stopSoundAnimals() function has been created which searches in sounds all the elements having the empty string and executes the stop() command.

4. Global Aspects

4.1. Technical details and problems encountered

The scene takes a while to load, so a **loading screen** has been added. The time taken varies depending on the **connection** and the **specifications of the computer** used, all this is reported to the user on the loading page. A **loadingManager** has been used for the loading, in fact, as soon as the loading of all the elements of the scene is completed, the onLoad() function of the loadingManager will be entered, and after this, instead of the **loading message**, a button with the word "**play**" appears. If the user clicks on it the game will start[1].
Initially, the game was thought of in a more detailed way, using a larger tribune and inserting many more cheering animals. Also the number of trees was much higher than the current ones. The game has been designed on **powerful computers** and during the development of the project no problems have been noticed, however in the **testing phase** other users have found some performance problems: the game did not run well, it was jerky; the animations of the raccoons and the camera were very slow especially for those who did not have high-performance computers because of the methodology used for these animations [3.5]. On less-performing computers, **fewer frames translate into a slower animation**. To overcome this problem, taking as speed to simulate that obtained with **144 fps**, the number of frames of each animation has been reduced using the following formula:

$$(number\ of\ frames\ needed\ for\ the\ animation) * \frac{144}{(actual\ frames\ generated\ by\ the\ running\ pc)}$$

In this way if a computer generates **less than 144 fps**, the number of frames needed for the animation is multiplied to a number smaller than 1 thus **reducing the number of frames needed for the animation and making it faster**.
Another technique used to optimize performance was to **remove some the initial elements** from the scene:

- **Trees**. The number of trees has been halved;
- **Tribune**. The number of animals on the tribune has been reduced to a third and the size of the tribune has been reduced accordingly;
- **Shadow**. One of the elements that weighed most on the performance was the presence of **shadows**. However, it has been decided to give the possibility to the user to activate or not the shadows (if the user has a high-performance computer) through a **checkbox** in the main menu, because the game is much more pleasant with the shadows activated.

As mentioned above, due to the policies of the Chrome browser[1], a special Html page has been created [3.8.1]. It will take you to the main menu with background music playing.
An array of pairs contains the raccoon management information. Each pair is associated with a raccoon. This pair has as a first element the **root of the object** useful for moving it, as second an object that has as attributes the **raccoon's bones**, this to facilitate their **transformation**.
To manage the **explosion of balloons** following a **mouse click**, **Three.Raycaster** was used. Thanks to this, a **ray** starts from the point on the screen clicked on. If this ray **intersects** a balloon, it explodes, playing a random sound among three audio files. The balloons were created immediately in a position **not visible** from the camera to facilitate the code. **Later, when the balloon should spawn, it is moved**.

5. Levels

There are three difficulties:

- **Easy**. Other raccoons are **much slower** than Steve's maximum speed;
- **Medium**. Other raccoons **might be a bit faster** than Steve's maximum speed. You will have to pop balloons to win. Also, it will be more difficult to reach the maximum speed;
- **Hard**. Other raccoons **will be faster** than Steve's maximum speed. The race will take place at night and balloons will give a lower boost. It will be more difficult to click on balloons.

Difficulty	Easy	Medium	Hard
Initial speed[2] of other raccons	80	80	73
Bar speed increase[3]	5% + (0-15)%	1% + (0-6) %	1% + (0 - 6) %
Bar speed decrease	0.001 per frame	0.00115 per frames	0.00115 per frames
Balloon boost	0.2	0.2	0.17
Balloon speed[4]	50/17	50/17	50/10
Range of speed [3.5]	70 - 90	60 - 80	60 -80

6. Conclusion

The aim of the project was to make a **browser game** that meets the **requirements**. The project contains a lot of **hierarchical models**. All models have been **animated** in different ways, some of them in a more thorough way, e.g. the dinosaur. However, only the **raccoon** model has been fully exploited for its animation. **Torches** and **balloons** were also animated. Not all aspects of the code have been covered due to their breadth. However, the code is well **commented**. If you look at it, you will understand aspects that have not been covered. the game has a lot of potentials, and new game elements could be added. For example, new character racers could be added with their characteristics and abilities, different, longer routes with different obstacles.

[2] *Speed is the number of frames needed to complete the animation. The higher it is, the longer the time taken.*
[3] *Range from one value to another. The higher the bar percentage is, the lower the bar speed increase percentage will be.*
[4] *Speed expressed as distance covered / time spent.*