# Final Project 2020

**Mahammad Namazov** NAMAZOV.1904736@STUDENTI.UNIROMA1.IT
1904736
**David Esteban Imbajoa Ruiz** IMBAJOARUIZ.1922212@STUDENTI.UNIROMA1.IT
1922212

## Abstract

The **BearrdCrash** game was built in the Babylon.js environment which has very spectacular properties in order to build web games. Game has competitive scenario that the figure drives the car and must avoid obstacles for finishing the parkour in given time. In the game there are some hierarchical models, which were built in both environments such as blender and Babylon.js, and animations were implemented by using specific libraries. As it is mentioned above, the BearrdCrash game has non-complex scenario and specific models and animations in order to satisfy that scenario.

## 1. Introduction

This report will include explanation for coding part of the game, how were animations implemented, models were used for the project and user interface which allows user interaction. The name of the BearrdCrash is given based on the character figure, which is called BearBirdBuddy, and the well-known video-game Crash which has remarkably similar scenario. Babylon.js was used to program the scenario of the game, animations, some hierarchical models and user interface. Additionally, there are some models were built by using the graphic programs such as blender and 3DS Max. In order to mention some words about scenario, it can be said that, character should collect coins, avoid bombs and fireballs in order to finish the game in the given strict time. To sum up, BearrdCrash is the easier and simpler version of the Crash game, which was built by using **Babylon.js**.

## 2. Environment: Babylon.js

3D visualization and gaming have come a long way, from implementations required a lot of work, including third-party plugins, and so on. The results we found were always impressive, but often incurred enormous costs and required countless hours of behind-the-scenes work. However, the options we have today make this process much easier. Nowadays there are tools provided by JavaScript provides us with some great answers with WebGL such as Three.js and Babylon.js, which uses this engine to provide browser solutions for 3D viewing and gaming.

WebGL based on JavaScript and shader logic that is executed on a Graphics Processing Unit (GPU), allows the implementations for 3D graphics can directly. all those implementations are hard to handle or are independently implemented. In this scenario appear the two of the most flexible open-source libraries, Three.js and Babylon.js, come to the rescue.

Three.js and Babylon.js are JavaScript-based frameworks that handle the complexities of WebGL with ease and provide a user-friendly way of implementing 3D graphics. While Three.js was designed as a tool for general-purpose web animations, Babylon.js was designed to take a more targeted approach for web-based game development and uses an extensive physics engine for collision detection and other intricacies involved for 3D graphics, for this reason, we've chosen the second one. Where the possibilities are endless–from simulating an actual fluttering cloth to a simulation of an entire universe with a 360 view camera rotation. The rendering of landscapes and terrains are nearly perfect examples of the beauty of these frameworks and so on.

## 3. Design

### 3.1 Hierarchical models

Hierarchical models part of the report involves explanation and information about the design of the hierarchical meshes. There are multiple models such as BearBirdBuddy, Car, bombs, coins, speed amplification platform that were designed hierarchically. We will explain these models' design methods in this section of the report.

#### 3.1.1 CITY

The design of the city was made taking as consideration the game ”Crash Bandicoot”, then the city has a certain similarity with a cartoon world. The Complete environment is conformed, in the first instance by the river with a bridge as a direct child and a big rock that allows an impression of a floating world. finally, the last principal object is the floor, that essentially is the base of the model, then are derived other objects like the hills, also were created rocks in random positions and also near to the road that with the aim to be little obstacles for the racing car.

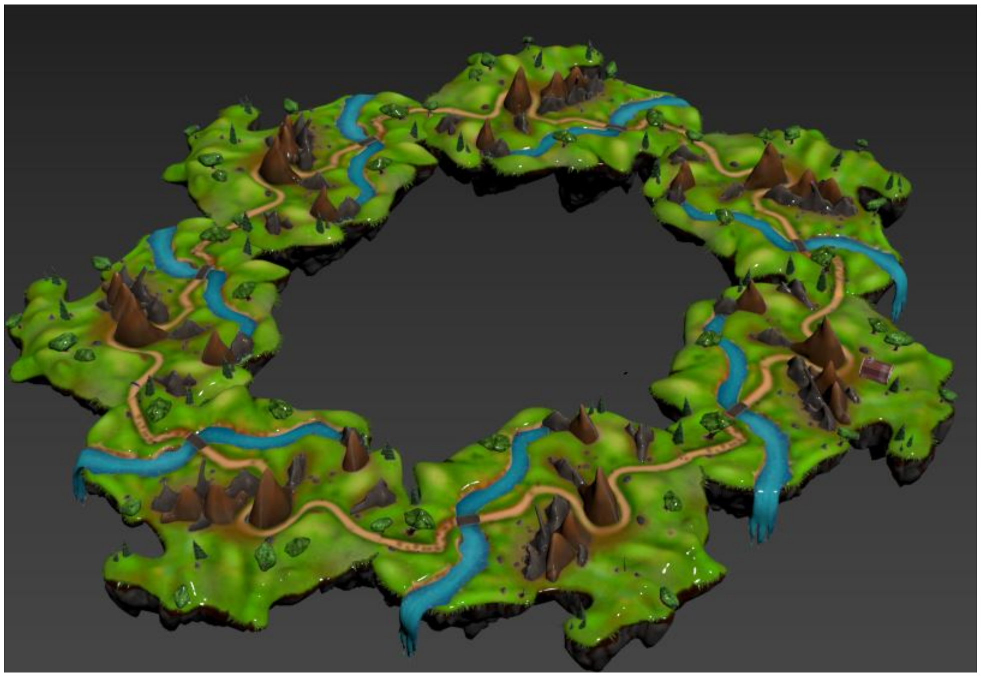


Figure 1: Final hierarchical city model, designed in 3Ds Max

The Floor has one last important element the trees that also has been located randomly and each tree has one or more leaves as children The hierarchical model can be seen in the figure 2 and a visual representation is shown in 1.

#### 3.1.2 RACING CAR

The racing car is also one of the most complex part implemented, the general hierarchical model can be seen in 4, where the tree parts ”Body”, ”Chassis” and wheels conform the same hierarchical model but due to the size of the final diagram these have been split independently in order to give a better explanation, as follows:

- Body: In the first place, we have to mention the Body skeleton that is used for the physics part and that will be explained later. also were used covering and defence parts to give a

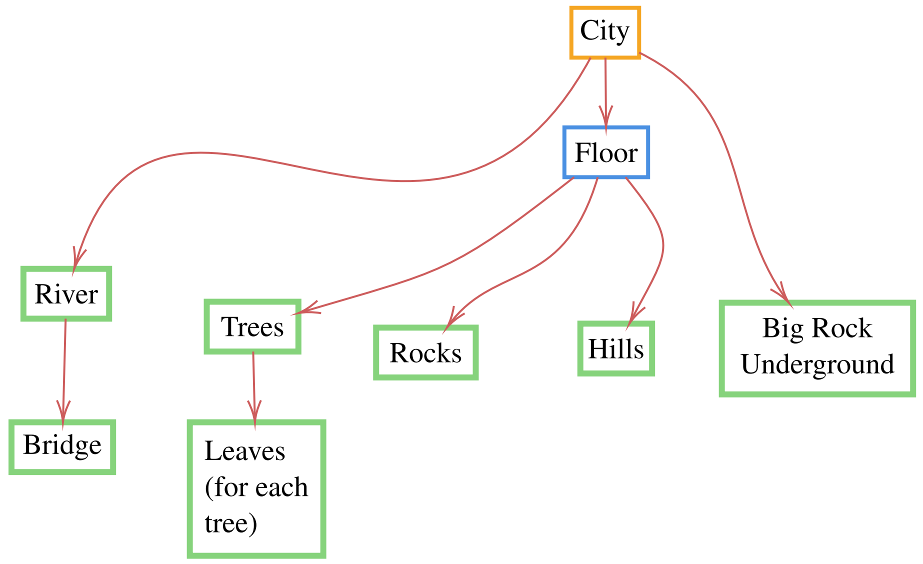


Figure 2: In the figure can be seen the hierarchical model of the city model implemented in this game.

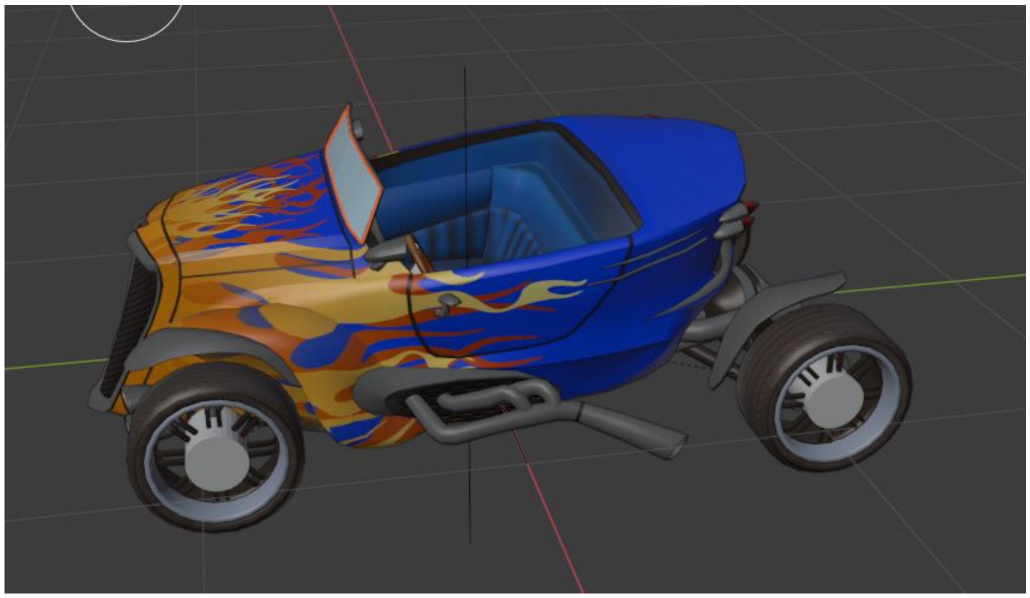


Figure 3: Final hierarchical racing car model, designed in 3Ds Max

better appearance to the car, together with a panel where are located a couple of clocks the hierarchical model can be seen in.

- Chassis: This part of the car is the most complex because here is integrated all specific part of a car, then we have axles, emblems, exhaust, headlights and even the Rear differential but maybe the most remarkable part is the steering wheel, the diagram can be seen in.

- Wheels : For each of the four wheels their invisible cylinders that are used in the physics part and also will be explained later. Each wheel has three hierarchical parts the first is the rim, its child is the tire and finally the support of the wheel. The diagram can be seen in.

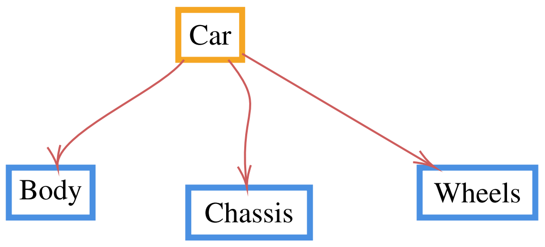


Figure 4: In the figure can be seen the main part of the hierarchical model of the Racing Car Model implemented in this game.

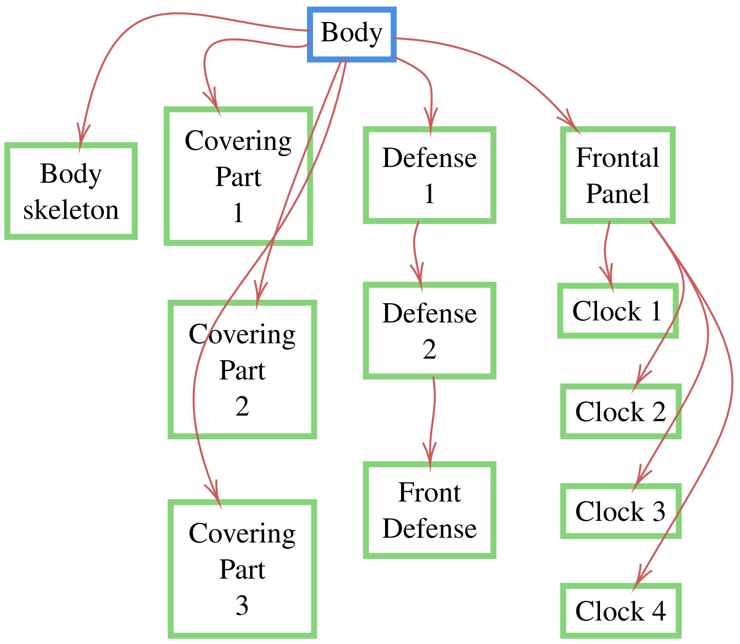


Figure 5: In the figure can be seen the Body part of the hierarchical model of the Racing Car Model implemented in this game.

### 3.1.3 BEARBIRDBUDDY

BearBirdBuddy is the main character of the game which is the car driver. The character was built by using Babylon.js. The main idea of making hierarchical model for the character was building it in compatible way for animations. All body parts are in the hierarchical model and the diagram of the hierarchical model is given in the Figure 8. This part of report will include all significant details about the BearBirdBuddy model. First of all, it has to be mentioned that the main mesh is Torso, which is the core mesh of the body and all part of Buddy are dependent with this in direct or indirect way. All head parts such as ears, eyes, nose and hat are children of the head mesh and hat disc is the children of the hat. On the other hand, each shoulder mesh (right and left) are parents of corresponding arms and arms are parents of corresponding hands. Additionally, in order to animate lower and upper body flexibly, we add lower torso which is directly connected to the Torso. Lower parts of the body are connected to the lower torso mesh with the same hierarchical design such as, shoes are children of feet, feet are children of lower legs which are children of knees and the rest are designed with this hierarchical mechanism. It allows us to animate BearBirdBuddy character's part independently and correctly. Hierarchy is important in terms of animations and parts are moving directly connected to the one main mesh which is Torso.

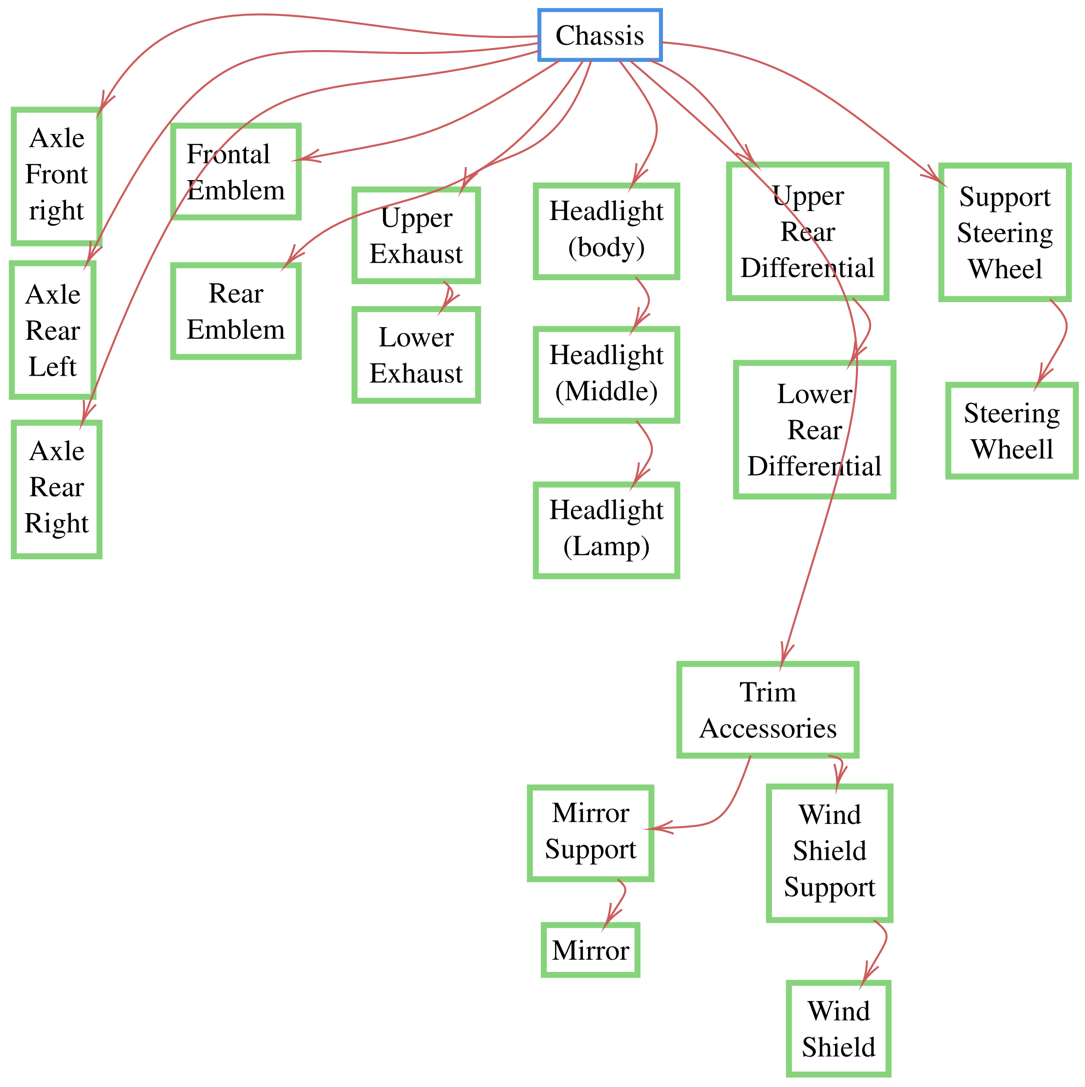


Figure 6: In the figure can be seen the Chassis part of the hierarchical model of the Racing Car Model implemented in this game.

### 3.1.4 BOMBS

Bombs are the main penalty effects in the game. While the car hits one of bombs, because of the explosion sight of the player is going to be closed for a while. It is also hierarchical model which involves 4 main parts. The main torso is the body of bomb which is sphere and it has bomb hood which was created by using cylinder. Additionally, the tail of the bomb was added which is parent of fire mesh and children of the bomb hood. Animation of that is working with invisible plains, because of the scene loop. The volume of the bomb and the car demand few scene loadings in order to activate the bomb. However, we need only one touch for the activation of the bomb effect. It has to be mentioned that, the torso of the bomb is the children of that invisible plain, because we need the stable plain and rotating bomb for game effects.

### 3.1.5 SPEED AMPLIFICATION PLATFORMS

The Speed Amplification Platforms are the reward mechanism that boosts the speed of the car while the car touches the invisible plane of the platform. The invisible plane is necessary for this part also

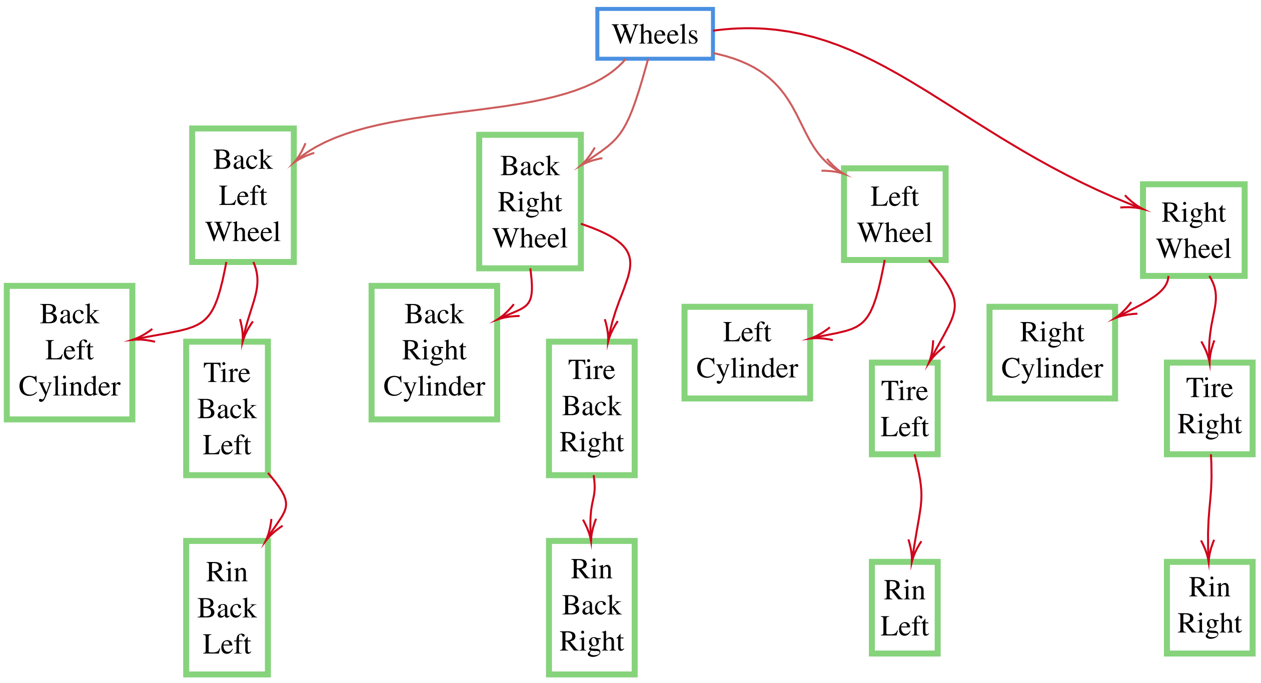


Figure 7: In the figure can be seen the Wheels part of the hierarchical model of the Racing Car Model implemented in this game.

because of the same reason that we have mentioned above while explaining the bomb meshes. It has to be explained that, while the hidden mesh of the car is intersecting the hidden plane of the speed amplification platform, it accelerates in the forward direction of the car. For instance, even if car hits the invisible plane while driving back, it will be accelerate in the forward direction of the car.

#### 3.1.6 COINS

Coins have hierarchical models, too but we see only the coins themselves but not invisible planes. We needed only one cylinder in order to generate multiple coins. Additionally, the coins have invisible planes in order to count when the car hits them. The Coin meshes are children of the invisible planes because we need stable plane and rotating coins in order to make fancy game effects. While car hits the coin, coin counter counts the number of the coins which were gathered.

### 3.2 No-Hierarchical models

#### 3.2.1 BUILDINGS

We've imported some buildings not-hierarchical in order to add some trim accessories, these have been put over the map and also works as obstacles for the racing car.

#### 3.2.2 FIREBALL

The fireballs are created from a sphere and then using the node material function provided by Babylon is possible to add and effect of the fireball as is shown in the figure. The fireballs are generated randomly depending on the car position and type of difficulty specified in settings by the user.

### 3.3 Lights

In this project were implemented 2 different types of lights:

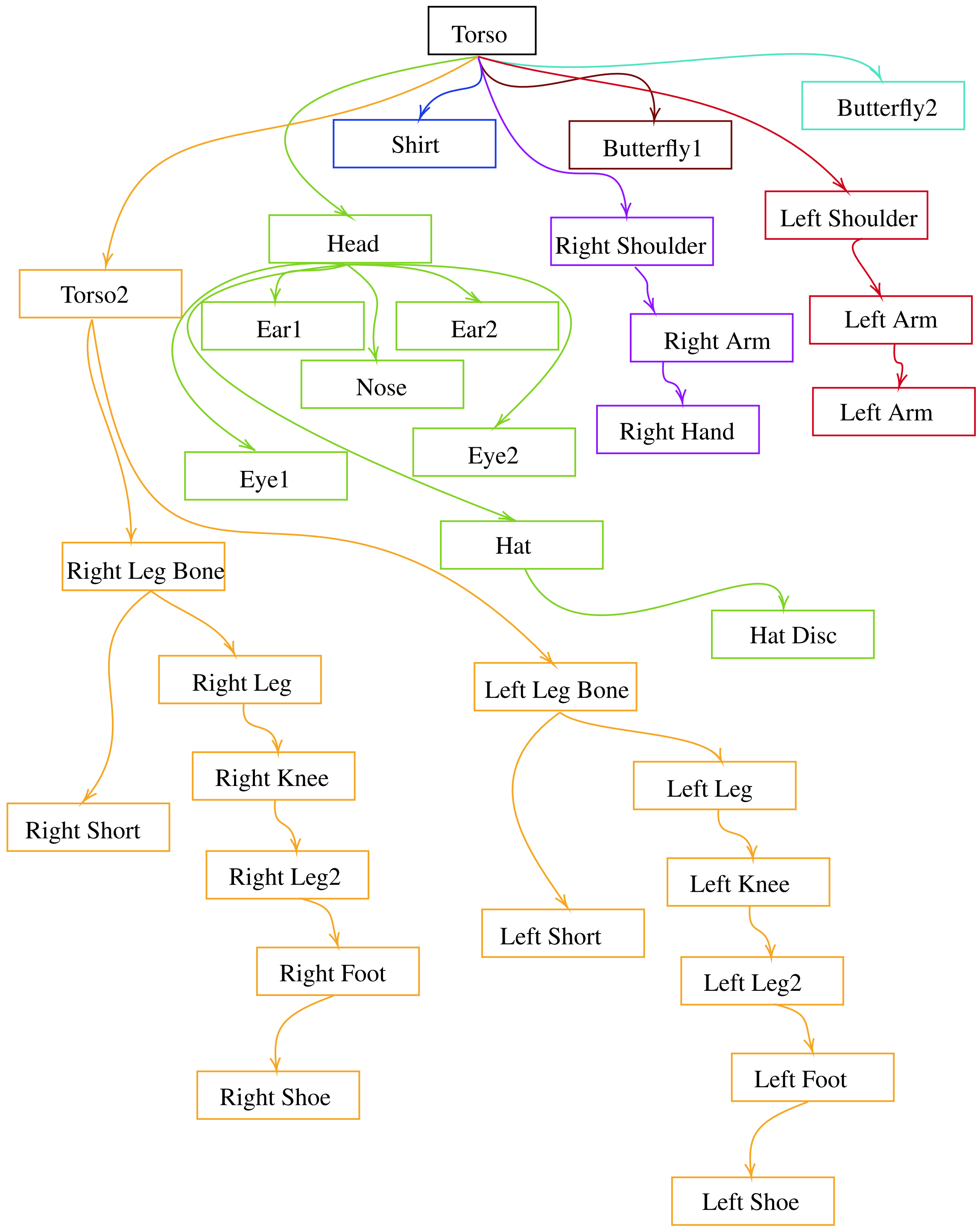


Figure 8: In the figure can be seen the hierarchical model of the character model implemented in this game.

- The first is a Hemispheric Light simulates the ambient environment light, so the passing direction is the light reflection direction, not the incoming direction.

- The next one, a directional light, was used to the implementation of the shadows and that will be explained in the next section. This light was also integrated together a sun model.

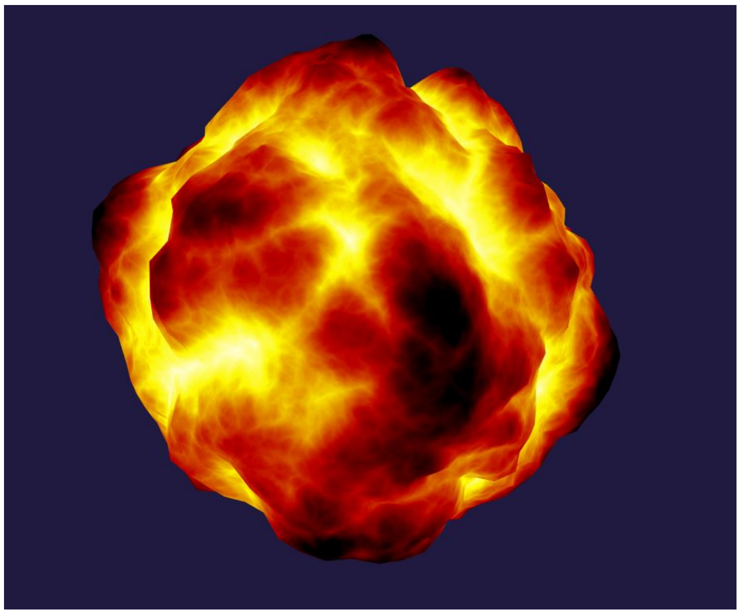


Figure 9: Fireball used for the game developed

### 3.4 Shadows

Babylon allows complete integration of shadows, for this, and with having to define a directional light, we've set up the shadows for the car and for the remaining objects, like the fireball. As a final step, the ground, the city in this case, was enabled to receive all shadows interactions, one result can be seen in 10.

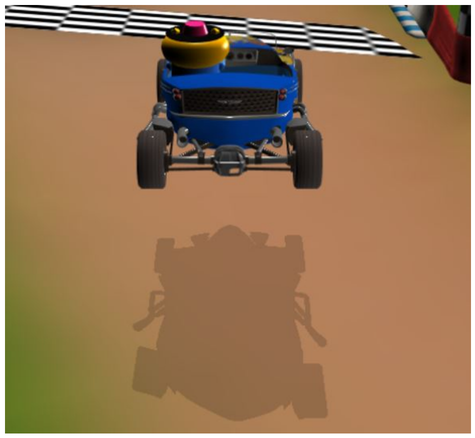


Figure 10: Example of the shadows implemented for the game developed.

### 3.5 Textures

The most textures were added using the software Blender and 3Ds Max, in fact in the repository can bee seen all images used to assign the textures. The textures have been chosen according to the cartoon style, it is notable for the entire city, for example, or the river and so on. For the car, the body and chassis have also the same style although for the body and chassis a little effect of metallic material were added.

## 4. Animations

This part of the report is added for explaining the animations that exist in the game. We are going to explain how to animations have been done in the code and the physics library implementation's affects on our game.

### 4.1 Physics

A physics engine is a software, a group of codes or a collection of objects and algorithms that provides functions for the simulation of a physical world, be it in games or other virtual reality application. The Engine used is Ammo JS it allows the movement of a bowling ball or how it collides with pins, It allows us to have a good representation of physics objects and the actions simulated in a realistic way.

The procedure to play the physics effect is in the first instance, enable the global gravity that will affect to all objects, and after this, We must convert all objects that should be affected by the physics properties. The ground and in this case the city defined with a mass equal to 0 in order to avoid that it going down. the Remaining objects converted were the car, to guarantee that the user can perceive a real impression in driving.

The Fireballs also were converted to physicist objects, in order to guarantees the collision of these with the car when the user set the difficulty in the hard mode, after this the fireball effects are added to each sphere.

### 4.2 Car animation

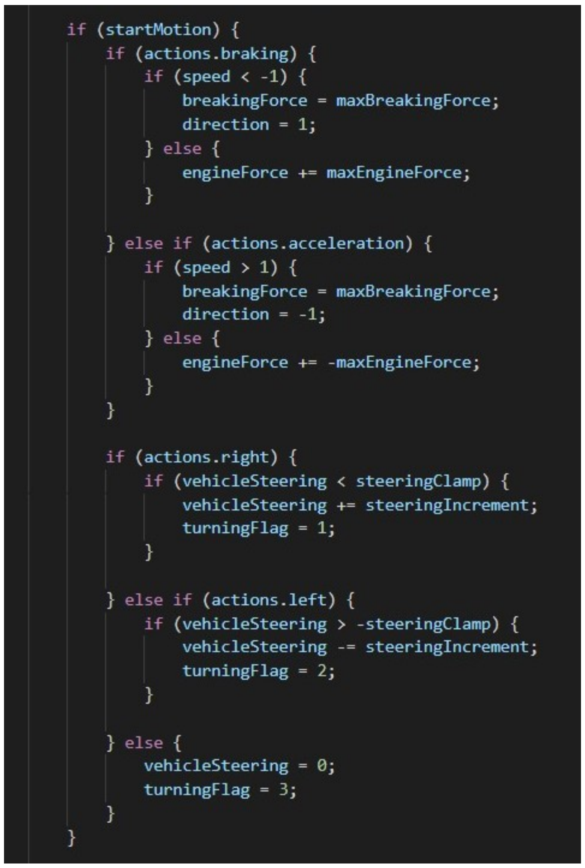
The main problem was car was not feeling the ground during the game. Fortunately, we solved this problem by implementing cannon.js library. Additionally, car animation is controlled by some animation flags in order to avoid confusions in the game. First, car cannot move until the player press key P after pressing start button on the start menu. Then the countdown with 5 seconds starts and game will start after 5 seconds. Notice that, unless car moves, timer will not be started. Then W, S, A and D keys will be activated in order to control the car. The Figure 3 represents how is the car controlled by users. Braking, acceleration, left and right are elements of the actions dictionary and are activated by pressing, S, W, A and D keys, respectively. It is seen from the Figure 11 that, by pressing A and D keys, turningflag gets different numbers such as 1,2 and 3 for turning right, turning left and resetting the wheel position. This part of the animation is shown in the Figure 12, which we implemented built-in rotation function of the Babylon.js and the reference of the rotation is taken as the car torso.

### 4.3 BearbirdBuddy animation

BearBirdBuddy animation includes only to turn the body right and left with respect to the local reference of the body. This rotation is activated by turningflag variable, too. The animation of the main character of the game is connected with the car animation. When car turns right, higher part of the body of the character tends to move to right and it is the same for turning left scenario for the car. Additionally when the turningflag is set to 3, the body rotation is reset to the initial position as well as the car's wheels. In other words, if A and D keys are not pressed anymore, the rotation of the wheels and BearBirdBuddy is reset.

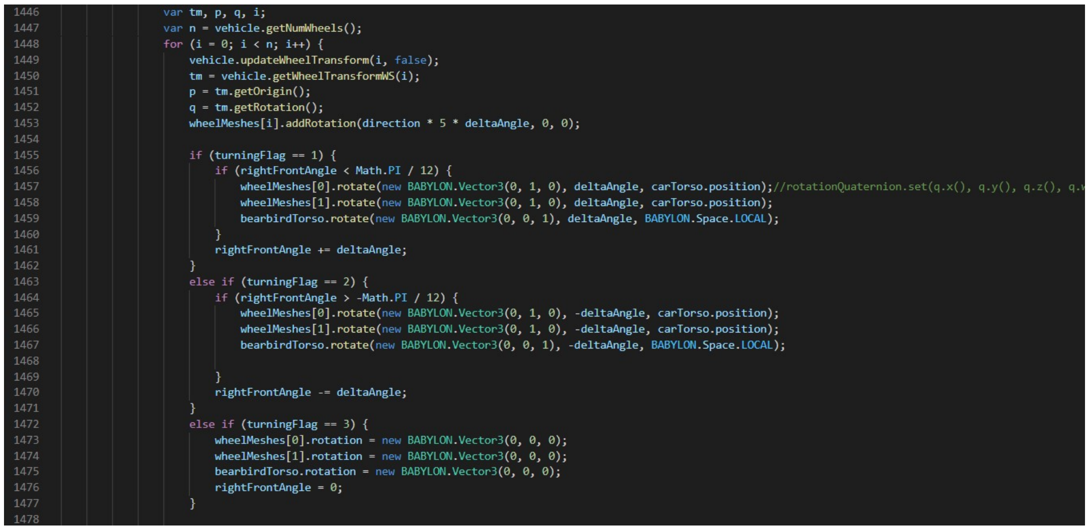
### 4.4 Bomb explosion

Before explanation bomb explosion it has to be mentioned that we made only one bomb mesh and put them in different places in the environment. While car hits the bombs during the game path, there will be bomb explosion effect. This part of the report, we are going to explain the working principle of the bomb explosion effect. The Figure 13 observes the function that is called during



```
if (startMotion) {
    if (actions.braking) {
        if (speed < -1) {
            breakingForce = maxBreakingForce;
            direction = 1;
        } else {
            engineForce += maxEngineForce;
        }

    } else if (actions.acceleration) {
        if (speed > 1) {
            breakingForce = maxBreakingForce;
            direction = -1;
        } else {
            engineForce += -maxEngineForce;
        }
    }

    if (actions.right) {
        if (vehicleSteering < steeringClamp) {
            vehicleSteering += steeringIncrement;
            turningFlag = 1;
        }

    } else if (actions.left) {
        if (vehicleSteering > -steeringClamp) {
            vehicleSteering -= steeringIncrement;
            turningFlag = 2;
        }

    } else {
        vehicleSteering = 0;
        turningFlag = 3;
    }
}
```

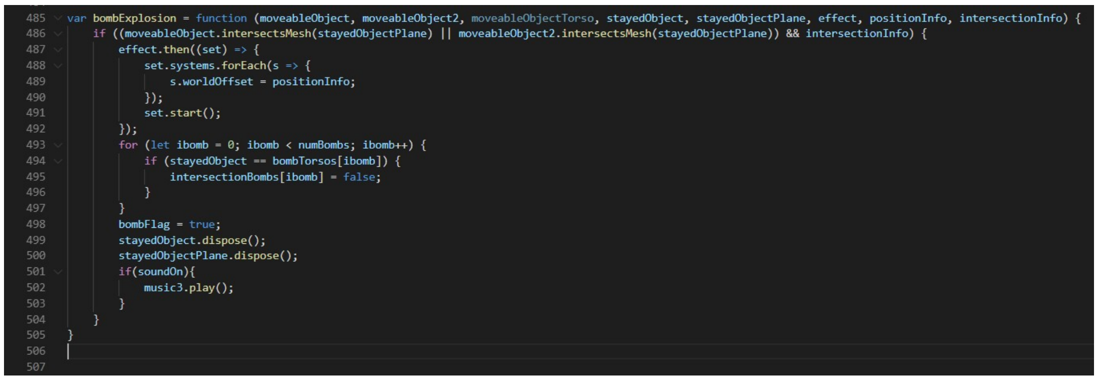
Figure 11: The animation of the car



```
var tm, p, q, i;
var n = vehicle.getNumWheels();
for (i = 0; i < n; i++) {
    vehicle.updateWheelTransform(i, false);
    tm = vehicle.getWheelTransformWS(i);
    p = tm.getOrigin();
    q = tm.getRotation();
    wheelMeshes[i].addRotation(direction * 5 * deltaAngle, 0, 0);

    if (turningFlag == 1) {
        if (rightFrontAngle < Math.PI / 12) {
            wheelMeshes[0].rotate(new BABYLON.Vector3(0, 1, 0), deltaAngle, carTorso.position);//rotationQuaternion.set(q.x(), q.y(), q.z(), q.w
            wheelMeshes[1].rotate(new BABYLON.Vector3(0, 1, 0), deltaAngle, carTorso.position);
            bearbirdTorso.rotate(new BABYLON.Vector3(0, 0, 1), deltaAngle, BABYLON.Space.LOCAL);
        }
        rightFrontAngle += deltaAngle;
    }
    else if (turningFlag == 2) {
        if (rightFrontAngle > -Math.PI / 12) {
            wheelMeshes[0].rotate(new BABYLON.Vector3(0, 1, 0), -deltaAngle, carTorso.position);
            wheelMeshes[1].rotate(new BABYLON.Vector3(0, 1, 0), -deltaAngle, carTorso.position);
            bearbirdTorso.rotate(new BABYLON.Vector3(0, 0, 1), -deltaAngle, BABYLON.Space.LOCAL);

        }
        rightFrontAngle -= deltaAngle;
    }
    else if (turningFlag == 3) {
        wheelMeshes[0].rotation = new BABYLON.Vector3(0, 0, 0);
        wheelMeshes[1].rotation = new BABYLON.Vector3(0, 0, 0);
        bearbirdTorso.rotation = new BABYLON.Vector3(0, 0, 0);
        rightFrontAngle = 0;
    }

```

Figure 12: The rotation of the car while turning right and left.

the loop of scene loading. However, bomb effect is activated in the function, if and only if the car hits the bomb's invisible plane. We used special bomb explosion effect which was built-in particle helper function of the Babylon.js and we can see the related part as "effect" in the following code snippets. The "worldOffset" part is important to be set, otherwise all bomb explosions are going to happen in the centre of the map which is not desired result for the realistic aspect of the game.



```
var bombExplosion = function (moveableObject, moveableObject2, moveableObjectTorso, stayedObject, stayedObjectPlane, effect, positionInfo, intersectionInfo) {
    if ((moveableObject.intersectsMesh(stayedObjectPlane) || moveableObject2.intersectsMesh(stayedObjectPlane)) && intersectionInfo) {
        effect.then((set) => {
            set.systems.forEach(s => {
                s.worldOffset = positionInfo;
            });
            set.start();
        });
        for (let ibomb = 0; ibomb < numBombs; ibomb++) {
            if (stayedObject == bombTorsos[ibomb]) {
                intersectionBombs[ibomb] = false;
            }
        }
        bombFlag = true;
        stayedObject.dispose();
        stayedObjectPlane.dispose();
        if(soundOn){
            music3.play();
        }
    }
}
```

Figure 13: Bomb explosion effect

## 5. User Interface

This part of the report explains how we realized the user interaction through the game. Implicitly, we will explain how to play game, and how to interact with start and pause menu.

### 5.1 Starting interface

While screen opens and game is ready, user will see start menu for the game. Start menu has multiple choices such as, start button, settings menu and help menu. These buttons are working with the clicking by mouse. The Figure 14 expresses the start menu and options that we added for players.

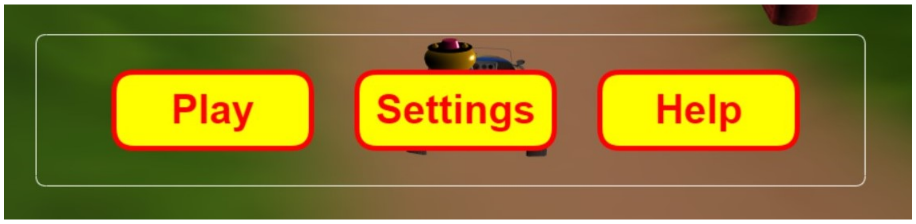


Figure 14: Starting User Interface of the game

### 5.2 Settings menu

Settings menu involves 5 main options such as light intensity, size of the screen, difficulty level of the game, day or night option and turning the sound on or off. The Figure 7 expresses the settings menu of the game. Difficulty is determined by the frequency of the fireballs around the stable range of the car. While we set it the minimum there will not be any fireball will disturb the car during the path. However, if we increase the difficulty there will be high frequent fireball attacks to the car. Additionally, difficulty option is disabled while game starts, and player will not be able to change the difficulty during the game anymore. Light intensity is changing by the slider that is given in the settings menu. The minimum value corresponds to dark screen which the player cannot see anything and the maximum value is 2 and it corresponds very low-contrast screen. Sound option gather all sound effects in one hand. In other words, if we turn it on, all sound effects and background sound will be activated. On the other hand, if we turn it off, then all sound related part will be disabled and there will be silence in the game, even if the bomb is hit by car.

The other option is screen size, which allows us adapt the game screen to any laptop device. We set width adaptive and slider changes the ratio which affects to fit the height of the game screen. Day or night option has not big affect on the game but it is another option that we added to the game.

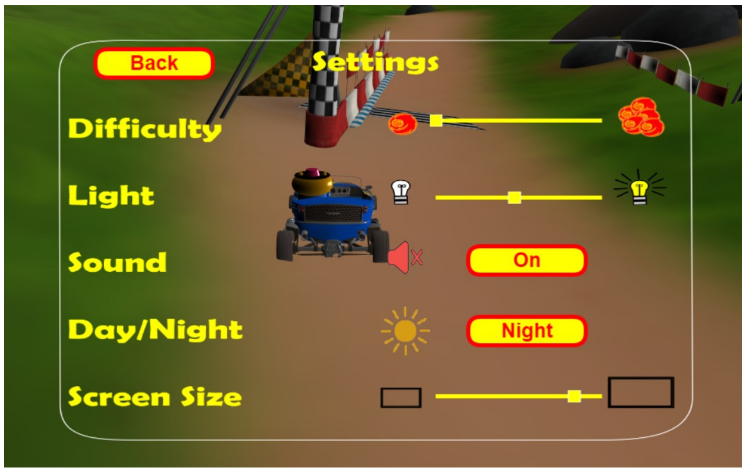
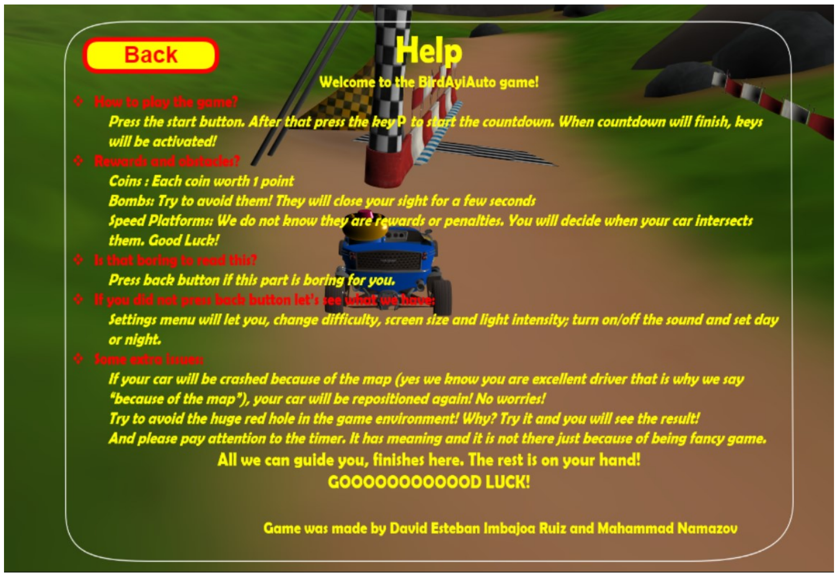


Figure 15: Settings Menu of the game.

While you set it day, the sky will be sunny and daylight will be on during the game. However, if the night will be set, then there will be night effect in the background of the game.

### 5.3 Help menu

Help menu is constructed in order to give some information about the game. It includes about the game starting scenario which is clue about how to activate the game. On the other hand, it gives some information about rewards and obstacles and also game scenario that player will see during the game. On the other hand, it also includes some unexpectable game scenario issues such as "Big Red Hole" in the map, crashing results because of the map. In order to give some expression, Figure 16 observes the Help menu.



Back

Help

Welcome to the BirdAyiAuto game!

- How to play the game?
  Press the start button. After that press the key P to start the countdown. When countdown will finish, keys will be activated!
- Rewards and obstacles?
  Coins : Each coin worth 1 point
  Bombs: Try to avoid them! They will close your sight for a few seconds
  Speed Platforms: We do not know they are rewards or penalties. You will decide when your car intersects them. Good Luck!
- Is that boring to read this?
  Press back button if this part is boring for you.
- If you did not press back button let's see what we have:
  Settings menu will let you, change difficulty, screen size and light intensity; turn on/off the sound and set day or night.
- Some extra issues:
  If your car will be crashed because of the map (yes we know you are excellent driver that is why we say "because of the map"), your car will be repositioned again! No worries!
  Try to avoid the huge red hole in the game environment! Why? Try it and you will see the result!
  And please pay attention to the timer. It has meaning and it is not there just because of being fancy game.

All we can guide you, finishes here. The rest is on your hand!

GOOOOOOOOOOD LUCK!

Game was made by David Esteban Imbajoa Ruiz and Mahammad Namazov

Figure 16: Help menu.

## 6. Conclusions

To sum up, the BearrdCrash game, which was built upon the car crash game idea. The game includes some hierarchical models such as BearBirdBuddy, city, car, bombs and some non-hierarchical models such as fireballs. The game has been built by using the Babylon.js by using some built-in functions and pyhisics library. The main idea behind of the game is the player has to finish the game route during the given time. The player also should gather the coins and must avoid the bombs, fireballs and stones. The game has been built as the course project of Interactive Graphics. Summary, the game was built by using Babylon.js upon the main idea of Crash game series.

## References