Interactive Graphics Luca Sannino 1542194

**FORMULA DRIVER DOCUMENTATION**

Formula Driver is a game developed with the Three.js library. Once the game is loaded in, the player is welcomed with a main menu showing a diorama-like scene where the user’s car is undergoing repairs at the pit lane. There, the user is able to change options regarding the difficulty of the game or some of its graphical settings, like the presence of shadows and fog, the resolution and the team the car belongs to. The player can also read about the premise of the game and the controls in the “About” section. When they are ready, the user can start the game by clicking the “Play” button.   
Once on the road, the player can move their car left or right in order to dodge incoming traffic, while gaining further speed in doing so, change the camera by switching from a cockpit to a third-person view and viceversa, and pause (and resume) the game.  
The game ends when the player crashes into an opponent, which sends the user’s car into a spinning animation, after which a game over screen showing the results is displayed on the screen, with a button that gives the player the option to return to the main menu.  
The goal of this document is to explain the technical details behind the game.

**SUMMARY:**  
**1. The Menu**  
 1.1 – init() pag. 2  
 1.2 – createMenu() pag. 2  
 1.2.1 – setScene() pag. 2  
 1.2.2 – modelsLoader() pag. 2  
 1.2.3 – tweakModels() pag. 3  
 1.2.4 – createMenuText() pag. 3  
 1.2.5 - assignButtons() pag. 4  
 1.3 – animateMenu() pag. 4  
**2. The Gameplay**  
 2.1 – createGameplay() pag. 4  
 2.1.1 – spawnNewCar() pag. 4  
 2.1.2 – setControls() pag. 5  
 2.2 – animateGameplay() pag. 5  
 2.2.1 – animateGamplayPlayer()  
 2.2.2 – animateGameplayRoad()  
 2.3 – gameOver()  
 2.3.1 – crashAnimation()

**1. The Menu  
1.1 – init():**  
Once the index.html page is loaded in, and after all the global variables are initialized, the first function to be called is *init(),* which in turn calls the functions *createMenu()* and *animateMenu()*.

**1.2 - createMenu():**It changes the title of the web page to “Formula Driver – Menu”, then calls the function *setScene()*. The next step, then, depends on if the game still needs to load the models or if the player has already played a game and is now coming back to the main menu after a game over. In the first case, the function *modelsLoader()* is called; otherwise, the models are already loaded in and *tweakModels()* is called. The next two methods are then called in both cases: *createMenuText()* and *assignButtons()*.

**1.2.1 – setScene():**This method creates the basic scene in which we display the main menu “diorama”. We first create a THREE.Scene with a white background, to which we add a white THREE.AmbientLight, which lights the whole scene, and two THREE.PointLight(s) in two different locations, which light only select parts of the scene and are able to cast shadows. We choose two of them for better presentation purposes.  
We then create a THREE.PerspectiveCamera that looks at the center of the scene, and a THREE.WebGLRenderer, which is basically the canvas where the game is displayed on and that we append to the body of the .html file.  
The renderer has its antialias and shadowmap settings enabled, allowing it to show an image with clean lines and shadows, if enabled by the user in the menu options.   
The renderer has also a size that is equal to the inner width and height of the browser window, but since said parameters can change when the window is resized, *setScene()* adds a listener that calls the function *setResolutionRenderer()*, with the objective to resize the renderer should this event happen, while updating the camera aspect-ratio.  
*setResolutionRenderer()* can also lower the internal resolution of the renderer, while keeping its dimensions intact, by first halving the size of the renderer, and then doubling the size of its internal domElement, which is the canvas part of the renderer: this happens if the player enables a low resolution in the main menu in order to improve the performance.

**1.2.2 – modelsLoader():**All the models in the game, from the car itself to the pit crew, were made thanks to the editor available in the official Three.js website.   
The player’s car, in particular, is a complex hierarchical model with over 50 parts of various geometries, like THREE.BoxBufferGeometry, THREE.CylinderBufferGeometry and THREE.TorusBufferGeometry, to which we add the material THREE.MeshPongMaterial (useful for rendering shiny surfaces) to create their THREE.Mesh.  
All of these models are exported in a .json file, which we then load in the .html file with the THREE.ObjectLoader. Once the process is done (the loader gives the opportunity to keep track of current state of the loading process), the *tweakModels()* function is called.

**1.2.3 – tweakModels():**  
This function alters some aspects of the models that are loaded in, like hiding some objects that mustn’t show up in the menu, changing some aspects of their materials (like the colors and the roughness), and adding textures and THREE.BoxHelper(s).  
We load the former by using the THREE.TextureLoader and then we map them to the matching materials: textures include the logos that show up on the body of the car, a normal map that gives depth to the tyres and a bump map that gives a more realistic feel to the body of the car.  
The latter, instead, are invisible boxes that we add to the front of the player’s car and to the back of the opponent cars, and that are essential for calculating the collisions later during gameplay. We resize these boxes to be smaller so that the player can get away with some extreme passes that should have caused collisions, in a way to not make the experience over punitive and improve the flow of the gameplay at the same time.   
*tweakModels()* also calls the function *wheelScreenText()*, which adds a texture to the screen inside the wheel of the player’s car. This screen normally shows the current number of passes and speed, but since we are in the main menu, it will display the sentence ‘READY TO START’.   
In order to do that, we create a new canvas element that we won’t add to the body of the document: instead we extract from it the CanvasRenderingContext2D, which is a 2D visualization of what’s inside the canvas, and where we write the string that needs to be displayed on the wheel using our chosen font and color. Done that, we create a texture that represents this canvas and map it to the screen’s material.  
If the player has enabled shadows (the game remembers its settings when the user goes back to the main menu after a game over), *tweakModels()* will also call *setShadows()*, which makes some of the objects able to cast and receive shadows, and sets the shadows’ mapSize to 512x512: higher values give better quality shadows, but at the cost of a worse performance.  
We then add the models to the scene.

**1.2.4 – createMenuText():**  
This function creates all the menu screens that go on top of the scene, which the player can interact with to change options, read about the premise and the controls, and to start the game. It essentially adds a series of ‘div’ and ‘button’ elements to the body of the document, whose parameters (or at least most of them) are detailed in the style.css file.

**1.2.5 – assignButtons():**  
This function assigns click events to all the buttons defined in *createMenuText()*. The buttons assigned to the “Options” sub-menu have the objective to update the global variables so that the user’s preferences can be remembered across consecutive games, and call functions such as *setResolutionRenderer()* and *setShadows()* to apply the graphical settings requested by the player.

**1.3 – animateMenu():**  
Once the entire scene has been created, *animateMenu()* simply rotates it around the Y axis, then requires an animation frame on *animateMenu()* itself, so that the canvas can be updated at each frame, and finally renders the scene using the renderer.

**2. The Gameplay  
2.1 – createGameplay():**  
This function sets the title of the web page to “Formula Driver – In Game”, stops the spinning animation on the menu and removes all text boxes, hides those objects that needed to be shown only on the previous screen while making visible the ones that are necessary now, and changes some rotation and position parameters on the camera and scene, whose background’s color we change from white to light blue.   
If enabled by the user in the options, a gray THREE.Fog is also added to the scene.  
Then the starting speed of the user’s car is initialized according to the chosen difficulty setting, while a timer that calls repeatedly the function *spawnNewCar()* is activated.  
An ‘div’ element that shows the score is added to the document, while we update the text on the wheel’s screen to tell the same score with *wheelScreeenText()*: of course, for now, this score is 0.  
*createGameplay()* then finally calls the function *setControls()* and *animateGameplay()*.

**2.1.1 – spawnNewCar():**  
This function initially checks if the game has been paused: in affirmative case, it doesn’t do anything. Otherwise, it updates an internal timer: if this timer has reached its max value, which changes constantly during the game since it depends on the current difficulty settings, then it is set back to 0 while a new opponent car is spawned on the far end of the road at a random position and with a random color.   
When we load the models in *modelsLoader()*, there’s actually only one opponent car, so in order to spawn a new car, we need to clone that object.  
It must be said that this function adds a drawback to the whole project: since *spawnNewCar()* is called at every interval, constant slowdowns reduce the rate at which it is called, which consequently reduce the rate at which the internal timer is updated, which in turn delays the creation of new cars. In order to fight this back, the game offers options to disable some graphical features, like shadows, and lower the resolution.

**2.1.2 – setControls():**  
This function adds listeners for keyboard input: pressing ‘a’ or ‘d’ updates a global variable that conveys the direction in which the player wants to turn to, while releasing them means the players wants to return to a neutral direction.  
The player can also switch between the two camera-views by pressing ‘v’: when this happens, the THREE.Camera’s position changes to a new set of coordinates, while the ‘div’ showing the score is hidden if playing with the cockpit view, or made visible if playing in third-person view.  
Finally, the player can pause the game by pressing ‘p’, which updates the title of the document to “Formula Driver – Paused” and stops the request of new animation frames.  
Of course, the game can be resumed by pressing ‘p’ again.

**2.2 – animateGameplay():**