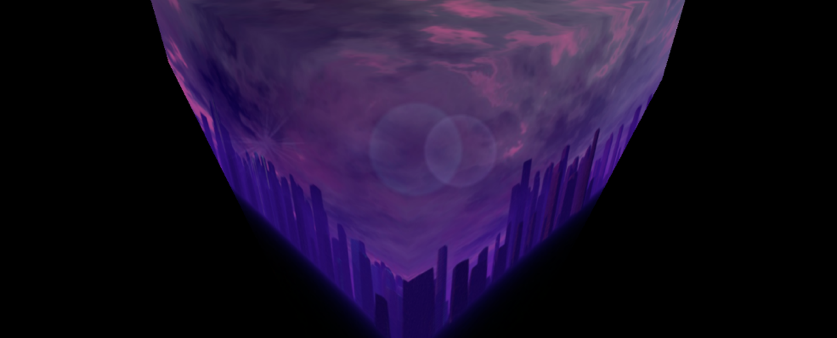
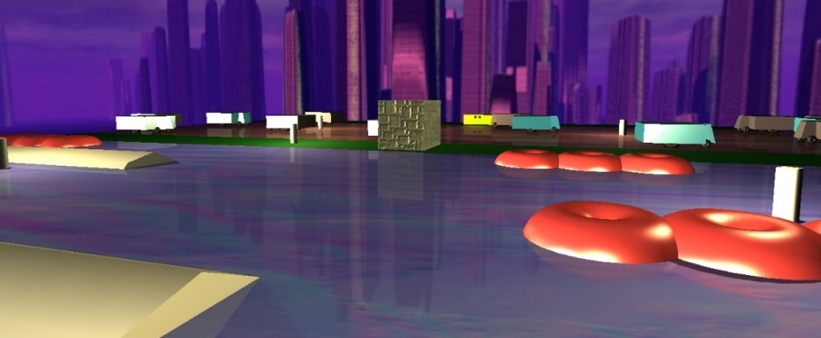
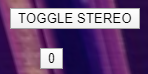
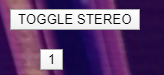
**Skybox**

We did not make our own shaders for the Skybox; instead, we just created a cube with double-sided faces, each with a different part the background, and added it to the center of the scene.

**Scoring System**

In the index.html file we added a new button (id = “score”) to the left side of the screen. That button starts with the value 0, and is updated with the current score value via document.getElementById(“score”).innerHTML whenever the frog reaches the other bank of the river, or when the player restarts the game.

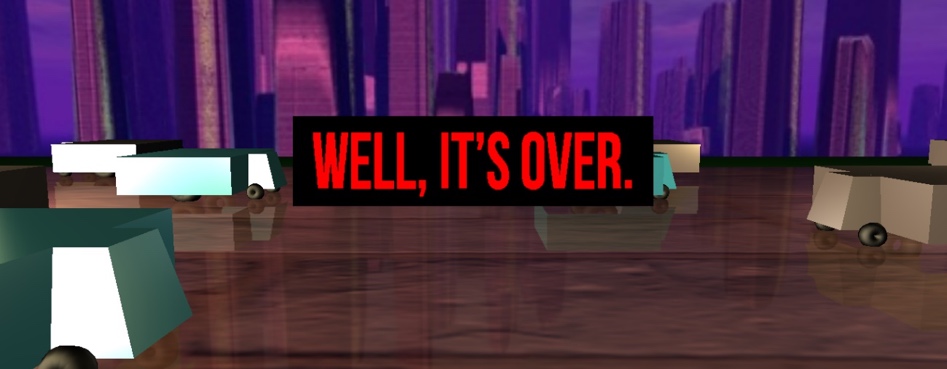
**Movement**

The frog moves when the player presses an arrow key. The game checks if the selected move is possible (i.e. if the frog would still be in the game area), and if it is, it increments (or decrements) the frog’s x or y coordinates to make the desired move.

Unlike the frog’s, every other moving object’s movement features acceleration. Each object’s speed is increased every 10 seconds by using a *setInterval* function, up to a maximum speed defined for each object (cars and trucks, turtles, and logs). Also, whenever an object reaches either the left or right side of the board, it is moved to opposite side of the board, to allow the reutilization of said object and save on memory and processing capacity.

**Collisions**

The collision system checks whether there are any collisions between the frog and the moving cars and trucks. It does so by checking if their spherical bounding boxes overlap. If they do, there is a collision, the frog loses a life (or the game if it has no lives left) and returns to the initial position. The same system is used to check whether the frog is on a turtle or log, and if it is, it moves along with it while they are still colliding.



**Stereo Viewing**

Upon pressing the TOGGLE STEREO button, the game enters fullscreen mode with 2 cameras, and the frog starts moving to the right on its own, until it reaches the right limit of the board, when it starts to shake a bit. The 2 cameras follow the frog’s movement by using a lookAt.



**Cameras**

A picture containing indoor, building

Description automatically generatedA picture containing indoor, game, object

Description automatically generatedA picture containing indoor, room, light, wall

Description automatically generatedIn the main file we initialize three cameras, one OrthograpicCamera, and two PerspectiveCameras, one above the frog and the other behind it, we change between this cameras with the keys ‘1’, ‘2’ and ‘3’, and we defined a function updateFollowCamera to actualize the camera 3 so that is always following the frog.

**Lights**

A picture containing indoor, monitor, light, dark

Description automatically generatedA picture containing television, indoor, monitor, screen

Description automatically generatedA picture containing indoor

Description automatically generatedIn this game we have one DirectionalLight set in the position (200,200,10) that we can enable or disable with the key ‘N’. There are 6 PointLights across the board that can be enabled or disabled using the key ‘C’ and we have one Spotlight in the frog, that is always in the front of the frog, we can enable or disable this light with the key ‘H’. Every light was done using Threejs.

**Textures and Bump-Mapping**

To implement the textures we defined a vertex shader and a fragment shader in the index and then in the js files we defined the uniforms were we send the textures, bump-map, the color components and the fog flag. We can get variables like the normal, the position and the light position in the index file to the shaders so we do not have to send them like in the opengl.

A close up of a brick wall

Description automatically generatedA picture containing indoor

Description automatically generated

**Planar Reflections and Planar Shadows**

In order to the implement reflections in this project we used a reflector from a git project and placed it under the table and then in the properties of the table, using Threejs, we defined an opacity of 0.85 we put transparent to true and used a blendSrcAlpha of 0.95 this way we are able to see the reflection of the objects in the reflector which gives the impression of a reflector floor.

For the shadows we used commands from the Threejs, so in every light that we created we used the attribute castShadow = true, so that every light casts a shadow, then if we pressed the key ‘L’ we activated the attribute receiveShadow of the table so that it receives the shadow of the objects casted by the lights.

A picture containing indoor, table, wall

Description automatically generated

**Billboard**

A picture containing indoor, table, room, wall

Description automatically generatedA picture containing indoor, computer

Description automatically generatedTo implement the billboard effect we use the function update() where we make certain that the billboards are always looking into the camera, implementing the Billboard Cylindric effect.

**Particle Effect**

The Particle Effect in our app is used when we win the game.

All particles have the same position, acceleration, life, fade and color but different velocities to simulate fireworks.

Then we draw the particles in a loop and for each step we update the position and its opacity.