Project Report

Interactive Graphics

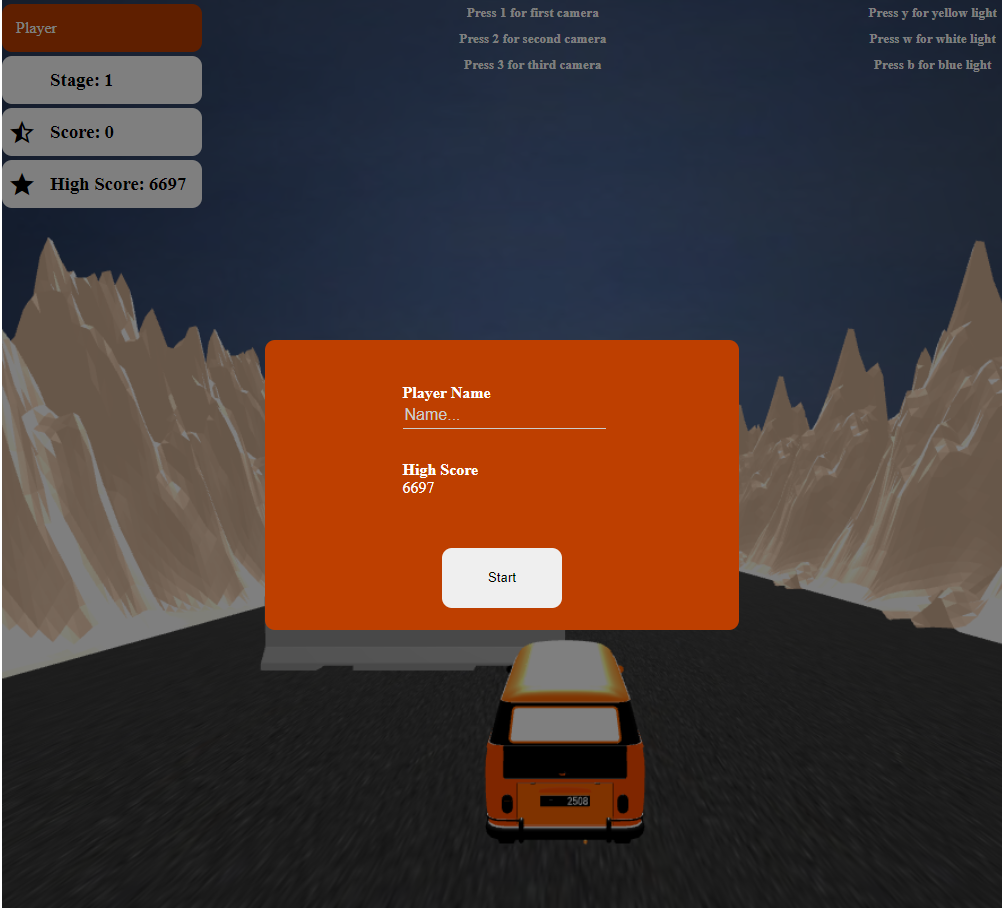
Car and Obstacles

Instructor: Prof. Shaerf Marco

Hamid …

…

June 26, 2021



# Technical Presentation

Moving the car through obstacles is a simple and fun game. In this game, the car, which controls the left, right, up and down directions, passes through obstacles quickly and if it encounters an obstacle on the way, it loses the game.  
The faster the car moves, the more points the player gets.

Game rules and logics have been developed with JavaScript language. The game is in 3D space and is consist of road, mountains, car and obstacles. This game includes hierarchical models, animations, textures, lights and user interactions to make changes the environment.

# Environment and libraries

I used these libraries:

* Three.js: Used to create and display animated 3D computer graphics in web browser. All the game rules and movements happen in this engine.
* OBJLoader, EasePack and GLTFLoader: These libraries used to load downloaded assets (board, checkers and dice) from files to game scene.
* Tween, TweenLite: Used to smooth and ease animations.

Environment has multiple key events to control lights, changing camera position and moving car.

The model used are as follows:

* Car (Model from free3d.com)
* Mountain (Model from free3d.com)
* Sky (Image from Google)
* Ground (Image from Google)
* Obstacle (Model from free3d.com)
* Start and End Line (Image from Google)

# Technical aspects

The game is in 3D space and the main scene has been initialized in three.js editor. The initial state includes 1 car (includes 2 spot lights), 3 obstacle, a lot of mountains in both sides, a road, a sky, two lights and one perspective camera; the obstacles generate dynamically inside the game. Now we will explain from creating assets dynamically to starting and playing game in details. For simplicity I used many global variables and decreased the functions arguments.

Game Cycle

1. Initializing game
2. Start
3. Reset
4. Move
5. If the car hits an obstacle, go to 7
6. If the car crosses the end line, go to 3
7. Save high score and show the result
8. Initializing Game  
   Initializing the game starts with the game\_init() which generates the main scene (prepared scene will be added to this scene), generates game renderer, generate a raycaster to detect car collision for moving checkers and loads the prepared scene (GLTF file using GLTF loader library). Important lines are:

set\_events(); //registers key events

scene\_main = new THREE.Scene(); //creates scene

aspect = WALLW / WALLH; //calculate the aspect of scene

var fov = 70; //field of view

camera = new THREE.PerspectiveCamera(fov, aspect, 1, 10000); //creates new camera

//creates new renderer

renderer = new THREE.WebGLRenderer({

    antialias: true,

    alpha: true

});

renderer.setSize(WALLW, WALLH, false);

var loader = new THREE.GLTFLoader();

loader.load('assets/objects/scene.gltf', function (scene) { //loads scene

    game\_scene\_init(scene); //initializes scene

    game\_stage\_reset(); //puts everything in its position

});

1. Start  
   Once we initialized the game, we will initialize the game scene, this includes finding the objects and storing them in variables, also we set the road(ground) texture to the downloaded asphalt image.

The second step is that we should set the car lights targets, which can illuminate the front of the car. We do this by adding to objects as target for the car lights. These two objects are children of car, and when car moves, they are moving in front of car too. Therefore, car headlights always shine directly in front of the car. The positions of the two light\_right\_target and light\_left\_target are obtained by trial and error and are already set in three.js editor.  
Also, we set the car to be the target of two main lights.

light\_1.target = car;

light\_2.target = car;

//

light\_spot\_right.target = light\_right\_target;

light\_spot\_left.target = light\_left\_target;

1. Reset  
   In this step we reset everything to initial state, creating obstacles, start to calculating score, capturing car collisions and moving the car.  
   First at this point, we set everything to its starting point, like car position, camera position, car tires rotation and etc. Also, we set all moving variables like speed, wheeling and etc. to their first state which indicates a stopped car.  
   Then we delete old obstacles and create the new obstacles. The number of obstacles depends on the stage we are in.

let n = stage \* 3;

we have three type of obstacles, type 1 the smallest, type 2 medium and type 3 the largest. The probability of using these obstacles is as follows:

let rnd = (Math.random() \* 70) + (stage);

if (rnd < 40) {

    obstacle\_new = obstacle\_1.clone(true);

    distance\_z = 7;

} else if (rnd < 60) {

    obstacle\_new = obstacle\_2.clone(true);

    distance\_z = 5;

} else {

    obstacle\_new = obstacle\_3.clone(true);

    distance\_z = 3;

}

As you can see, we used the stage in our calcs, this means, the upper stages, more larger obstacles.   
Because obstacle sizes vary, their starting point is also important across the route, so they do not get off the road. For this reason, we have set the maximum distance that the obstacle can be from the right side of the road.  
we divide the road length to n pieces, and then multiplying it to a random number to find the next x position of obstacle.

let x\_to\_add = (Math.random() \* (distance.x / n));

pos = {

    x: pos.x - x\_to\_add,

    z: obstacle\_min.z + (Math.random() \* distance\_z)

};

* "distance" is width and length of the road.

After creating obstacles, we start the game main logics:

game\_animate();

iv\_score = setInterval(game\_calc\_score, 500);

iv\_collision = setInterval(game\_check\_colision, 5);

iv\_move = setInterval(game\_handle\_move, 10);

We call game\_animate function to render the scene:

if (!ingame) return;

if (paused){

    setTimeout(game\_animate, 100);

    return;

}

camera.lookAt(vector\_3\_add(new THREE.Vector3(car.position.x, 0, 0), camera\_to\_car\_diff));

game\_render();

TWEEN.update();

requestAnimationFrame(game\_animate);

If player be in the end of stage, we set the pause variable to true, in order to resetting stage, so we return after 100ms to render the scene to check if reset\_stage is done or not. Each frame we set the camera target to car position. Also, we call the tween update function to update our vector changes to have the animations. At the end, we request next frame animation.

1. Move  
   Movement is controlled by four arrow keys. Left, right, down and up.

**Accelerating**: When we hold the up-arrow key, we increase the speed variable by the acceleration (a) factor, note that we are not interfering with the movement of the car here. We handle car movement in game\_hande\_move function.

**Brake**: When we hold the down-arrow key, with double of the acceleration, we reduce our speed (a \* 2).

**Idle**: When we do not press any key, we go into idle move, which we reduce our speed with half of the acceleration (a / 2).

**Keep Left or Right**: When we hold the left-arrow or right-arrow key, we change the car z position speed\_wheel units to the sides in 0.3s. This happens with tween updates. Also, we rotate the car tires to the left or right during the operation (tires\_rotate(dir)).

Now we explain game\_hande\_move function:

tween = TweenLite.to(car.position, 0.05, {

    x: parseFloat(car.position.x - speed).toFixed(3),

    ease: Elastic.easeInOut

});

We move the car position with pace of speed. TweenLite to function makes this change in 0.05s so we see the ease movement of car. Also, we move the camera behind the car with this animation.

1. Collision  
   We keep the obstacle mesh in obstacles variable, and every 5 ms we check the car collision with these obstacles as follow:

var ray = new THREE.Raycaster(car.position, directionVector.normalize());

var collisions = ray.intersectObjects(obstacles);

if (collisions.length > 0) {

    for (let i = 0; i < collisions.length; i++) {

        if (collisions[i].distance < 1) {

            game\_finish();

            return;

        }

    }

}

1. Stage Finish  
   If car crosses the end line, we reset the stage and game to start next stage. Here we set the paused variable to true, to reset the stage. This happens with game\_stage\_function:

if (paused) return;

paused = true;

setTimeout(function(){

    stage++;

    document.querySelector("#header\_stage").innerText = stage;

    game\_stage\_reset();

}, 500);

setTimeout(function () {

    paused = false;

}, 1000);

1. If any collision detected, we stop the game and check the high score.

# User interactions

The main interactions of player with game are the car movements. Moving car is implemented by key down and up events. We set the movement start with pressing down the key and end them when player release the key.

function \_keydown(e) {

    if (!ingame)

        return;

    if (paused)

        return;

    if (e.key == "ArrowUp") {

        is\_accelerating = true;

    }

    if (e.key == "ArrowDown") {

        is\_braking = true;

    }

    if (e.key == "ArrowLeft") {

        keeping\_left = true;

    }

    if (e.key == "ArrowRight") {

        keeping\_right = true;

    }

}

Other interactions are changing camera position and color of lights. We capture player pressing prepared keys and perform their actions.   
Camera position changes by tween animation to give player better experience.

function camera\_position(location = 0) {

    if (location == 0) {

        camera.position.x = camera\_position\_init.x;

        camera.position.y = camera\_position\_init.y;

        location = 1;

    }

    let z = 0;

    if (car)

        z = car.position.z;

    if (location == 1) {

        tween = TweenLite.to(camera.position, 1.5, {

            z: z,

            ease: Elastic.ease

        });

    } else if (location == 2) {

        tween = TweenLite.to(camera.position, 1.5, {

            z: z + 2,

            ease: Elastic.ease

        });

    } else if (location == 3) {

        tween = TweenLite.to(camera.position, 1.5, {

            z: z - 2,

            ease: Elastic.ease,

        });

    }

}

We have three color for lights, which player can use, blue, yellow and white as the default color.

function lights\_change(choice) {

    switch (choice) {

        case "blue":

            lights\_blue();

            break;

        case "white":

            lights\_white();

            break;

        case "yellow":

            lights\_yellow();

            break;

    }

}

function lights\_blue() {

    light\_1.color = new THREE.Color(0, 0.06, 0.55);

    light\_2.color = new THREE.Color(0, 0.06, 0.55);

}

...

# References

<https://github.com/tweenjs/tween.js/>

<https://free3d.com/>

<https://github.com/zhbhun/easepack>

<https://github.com/johh/three-gltf-loader>

<https://github.com/Bly7/OBJ-Loader>

<https://github.com/greensock>