CAP 4720 Computer Graphics Final Project Report

Project Explanation: F-Zero is a single player racing game set in the far future where the cars have no wheels and hover, steering more like an airplane using vectored thrust and rutters that interact with the air than like a car which uses contact forces between the ground and the tires. The first game released in 1990 in Japan, and 1991 in North America and was made in order to showcase a faux-3D effect of the Super NES named Mode 7. This technique is somewhat similar to the cubemap skyboxes we have made before and allows for making a 3D background using 2D assets, which will be crucial to making a 3D game in a month without a game engine. The project I am proposing is to recreate and update the original F-Zero with 3D models replacing many of the 2D sprites the original game used.

How to run the project: Go to the observable link:
https://observablehq.com/@jastevens/cap-4720-project-final-version-additional-lighting-track-b

Project Controls:

- Click the button named SPACE under the title screen graphic
- 'A' and 'Z' to go forwards and backwards
- 'K' and 'L' to turn left and right (Make sure you are also pressing 'A' or 'Z' to rotate car)
- IF NOT WORKING, PLEASE CLICK ON THE GAME WINDOW

Project Visuals:
https://youtu.be/ldQ4FBYBu9c

Project code and functionality:

Feature 1 Player Model: Added the car that the player controls. I used the model rendering and texture mapping techniques that were shown in class and in previous assignments. The model moves using matrix translation that was also used in previous assignments.

Feature 2 Player UI: Used dat.gui api, a lightweight controller library for javascript. It allows you to easily manipulate variables and fire functions on the fly.

Feature 3 Physic: Nothing really special was added, just simple trial and error was made to get the correct numbers such as score, health regen, damage

Feature 4 Game Obstacles/Objects: I used tween.js for the game obstacle objects' movement animations. For the plate objects they move up and down, and for the rock objects horizontally.

Feature 5 Pit Zone Interaction: Changed shader outcolor based on fragpositions y value. Used cars x and z position to determine if the car is in the pit zone. Lower UFOs y value if the car is in the pit zone.

Feature 6 Track and Background: Used the skybox, texture mapping, and orbit camera as used in assignments. Used the transparency of the .png image format to make portions of the track quad transparent in the fragment shader. One quirk of doing this is that the transparent object must be rendered last in order to prevent the depth test from omitting things behind the transparent object.

Feature 7 CPU Racer: Used tween.js for animation purposes.

Feature 8 Additional Lighting (Bradley Vanderzalm): I implemented the nighttime skybox and applied a diffuse, specular, and ambient lighting on the car and spectators during night mode. I also checked the car's position to determine if a light effect should be applied.

Feature 9 Spectator Area: Implemented a custom function for populating the audience stands next to the track's starting line with alternating Rayman and boy models. It takes a viewMatrix, angle and scaleMatrix, and feeds these inputs to the render functions for the models, which alters their modelMatrices to put them in the right place on the bleachers.

Feature 10 Main Menu (Marcus Ford): After the rendering and canvas section of the code was developed, I edited the parameters of the while loop used to constantly update the game to activate and stay on after clicking the button "SPACE".

Feature 11 Particle Effect: Implemented particle and rocket classes. The particle class is one that is being called on which then assigns some kind of parameter to an attribute such as velocity, position. While also giving that particle some parameters like gravity and size. It then updates these parameters over time, and finally renders them with some kind of structure like a circle or square. The rocket class calls on this particle class to give it some value and then pass ``it on to the explode function which makes these values random on the velocity and position. This is then rendered onto the kind of shape that we want. In this case we want an arc in a circle shape, so we use the createRadialGradient function that is available for 2d and give it the basic values that it needs. This is all pushed on the mutable arrays that are constantly updated and then displayed on the canvas.

Group Members Contributions In Detail:

Jaelys Madera (Feature 1: Player Model): In order to create the model that the player would control, I found a free 3D model of the F-zero blue falcon car with the texture. For my contribution, I placed the model in the scene with the proper texture. I then scaled and translated the model so that it would start at the starting line on the track. I also wrote the initial code to have the model be rotated and translated with the user input.

Norbey Torres (Feature 2 Player Input and UI overlay): Implemented the user interface with dat.gui to directly manage all parameters such as camera position controls, camera angle controls, model controls, and game statistics. To accurately represent real time values, dat.gui

was developed with animation control by implementing a browser refresh display rate using a while loop. For player input, event listeners were added to trigger whenever a key was pressed and reset whenever the key was released. With the event listener, keyboard event key codes were mapped to manage the dat.gui parameters.

Quanminh Nguyen (Feature 3 Physics): In the end I did not use any outside libraries and just used simple math and trial and error to code the physics. One of the fixes was to link up the camera so that when the car accelerates the camera also follows it since it is not physically attached to the car. The car is able to accelerate and decelerate depending what button is pressed and has a max speed like all real vehicles. Made it so that if the car collides with specific objects it will lose health and when it reaches zero, the game will stop rending the car. And if the car crosses the finish line, it will increase the score. And to regenerate health, the player needs to be in the green landing strip which will heal them.

James Stevens (Feature 4 Game Objects/Obstacles: I used a scifi-plate model for the charging zone objects, and also a rock object that I found for the game obstacles. To make the plate and rock objects move I used tween.js for the animation. I decided not to use textures for the objects, and instead used material colors, and lighting effects. When a player collides with the rock objects, the car model's speed is reduced.

Paul Vicino: (Feature 5: Pit Zone Interaction): Found a UFO model object and texture, then edited the shaders of the model to influence the color of the beam that was casted down from the UFO. Then created a toggle on the UFO that affected its height. This was necessary because the UFO needed to lower its height into the camera view when the player's car entered the pit zone area (the neon green section of the track close to the starting line).

Matthew Hubbs (Feature 6 Track and Background): Created a textured quad for the sea with a second textured quad for the track. Utilized transparency of the .png file format for the track texture in order to allow the track to appear floating in air above the ocean. Added a skybox using a cubemap texture found from a google image search, editing that image to fit nicely in the environment. This was the first feature to be implemented, first creating a moveable camera for the 3d environment. These three things together gave the impression of a 3d world due to the parallax effect. Initial prototype of only this feature alone available at this embedded link.

Bobby Pappas (Feature 7 CPU Racer): Used the drawn CPU car textures added from feature 1 and implemented tween.js to create a moving CPU car for the user to race against. Basic animations with a tween chain allowed for seamless movement of the CPU racer.

Bradley Vanderzalm (Feature 8 Additional Lighting): My contribution involved implementing a night time mode where I kept track of the car's position where if it entered a particular area a lighting effect would appear on the car's model. A lighting effect was also applied on the spectators. I also helped determine if the car was located inside the green pit zone where I triggered the UFO to come down and then gave the car additional lighting with a green specular color to give it a green reflection from the UFO's green light beam. As you go along the track

(during night mode) you'll notice the lighting on the car change to simulate passing lamp posts, if there was additional time and/or additional members of the team I would put futuristic lamp posts next to these areas to give the true effect. NOTE: the default mode is daytime and the lighting effects are only present in the night mode. Please click the checkbox below the gl canvas to enable night time.

Allexis Knight (Feature 9 Spectator Area): Found a bleacher model to serve as audience stands, then scaled and translated the model in various ways until finding the right position and orientation to place the bleachers next to each side of the track near the starting line. Then imported Rayman and boy models and created several functions to populate the bleachers on each side with Rayman and boy models. Stays rendered and present at the starting line throughout the race.

Marcus Ford (Feature 10 Main Menu): I studied the functions of observable for about a week and in the end I did not make it as flashy as hoped. The original plan was to have an image/gif switch out and begin displaying and rendering the game once you press space. In the end, I only could do those separately, and replaced the space button requirement with a literal button named space (we couldn't change it to start since the graphic menu was personally made, couldn't change the text in the image). At least I till managed to have the game start on the click of a button by editing the parameters of the while loop. Despite having to click on the screen to start, the player is naturally faster than the enemy anyways, so the slight delay does not matter.

Eduardo Bourget (Feature 11 Particle Effect): The particle effects in this case were used over two mutable arrays which basically means that they allow for a constant pointer to change members, which is useful for changing particles rapidly. Since the primary thing I was doing was explosions on rockets, I had two classes: the particles and rocket class. The particles class provided a way of giving the particle's some parameters and being able to render them in some kind of way. The rocket's class took that class particle and assigned the particles random parameters for velocity, size, gravity, and others. These were assigned to be able to explode in some kind of way. These are then rendered and then pushed on the arrays to be displayed. In this kind of way we are able to create an effect of a rocket going up to a certain height of the canvas and then exploding in a random place with some kind of desired shape, which in this case is a big circle arc with little particles. At the end they all disappear and the process starts all over again.