Project Report (12/7)

**Objective**

Our project was to design a game in which the user controls an object on a track surrounded by a “Dead Zone”. The player navigates the object through a course called the “Safe Zone” using the arrow keys to control movement. If the player should navigate into the Dead Zone, they will lose the game and have to restart. To win the game, the player must navigate the track until they reach the finish line, the red area of the scene

**Features**

The program will load in a given track, different tracks will have different difficulties, and then use the white space to generate a safe zone and the black space to create the dead zone.

The player model being used is a colored cube, the cube starts at the center of the scene on a starting platform.

The player controls the movement of the cube using the arrows on the keyboard, a more in depth description of controls can be found in our user manual.

Using color detection the game can be ended 2 different ways: if the cube touches the dead zone (black) the game will end in a loss and restart. If the cube touches the finish line (red) the game will end in a win and give the option to restart.

At the start of every game the camera will rotate around the cube to give the player a preview of the course.

The game also incorporates 2 sounds, the first is a starting sound that plays as the camera revolves around the player. The second sound will play is the cube touches the dead zone and loses.

**Implementation**

The game is built in the game.js file in the game source directory. The gameWrapper.html filecontains the html to create the website, canvas, and also includes the shaders. Several implementation details of the scene are described below. Our game contains two separate components that are connected via two separate draw calls in drawScene(). The following details are described for each of the two draw calls: 1. For track, 2. For player model

* Geometry

1. A 250 x 250 grid of vertices is used for the geometry, with normals and texture coordinates programmatically generated for each vertex.
2. A simple hard coded cube is created with 8 vertices, with no normals or texture coordinates, but colors assigned to each vertex.

* Texture

1. Texture coordinates ranging from 0.0 to 1.0 are assigned to each of the 250 vertices vertically and horizontally, so as to map a power-of-two texture perfectly over the surface of the grid. This texture information is what determines the appearance of the track, as well as win/lose conditions. If the texture of a pixel is white, then it is part of the safe zone, if it is black, it is part of the dead zone, if it is red, it is part of the finish line.
2. No texture implemented for this geometry, rather a vertex color is assigned to each vertex, so as to interpolate the color between vertices and achieve a colorful, rainbow look over the model

* Shaders

1. A complicated shader is used for this geometry, allowing for pixel perfect texture coloring, and lighting and positional computation for each pixel. Based on vertex position calculated, pixels with a position below 1.0 Y are discarded and not drawn.
2. An extremely simple shader is used to compute the appropriate transform for the vertices, and then the fragment shader interpolates between vertex colors to generate rainbow colors across the faces of the cube.

* Model/View Matrix

1. The mat4.lookat() function is used to generate a view matrix that looks at the players position from a distance a little ways back and above to generate a nice 3rd person viewpoint to navigate the course in.
2. The same view matrix for the first geometry is used for the cube, but in addition the matrix is scaled down to size, and then translated to its proper position in the scene.

* Shading/Lighting

1. Goraud shading is used for the track with ambient and diffuse lighting.
2. Goraud shading is used for the cube with no lighting calculations.

* Movement

Movement is handled by modifying playerPos. When an arrow key is tapped, the derivative values for X and Z movement are modified, i.e. the user tapped up, so xSpeed is increased by 0.001. When a frame is rendered, then the playerPos is updated by adding the xSpeed and zSpeed to the X and Z positions respectively. This is how win/lose is figured. The pixel beneath playerPos is looked up in the texture, and if playerPos is unsafe, or at a win pixel at new frame render time, then the game handles this.

Movement is handled similarly visually, by generating a new mvMatrix at each frame, in which it uses mat4.lookat() to make the camera look at the playerPos from back and up a bit. This same matrix is then used to keep the player model in the center of the screen, by translating the X and Z position of the camera by the playerPos offsets from (0,0), which in turn makes the camera always centered on the player model, in 3rd person view, above the correct position on the track.

* Initial spin effect

When the player restarts the level, the camera will spin around the player model to show a preview of the track. This effect was achieved by having an initialSpinAngle variable that is initialized to 0, and slowly approaches 360 by increasing programmatically. Before 360, the mvMatrix is rotated by this angle, and after 360, the mvMatrix ignores the rotation. This allows the camera to pan around the player model to show the whole track, but it only spins around once. If the initial spin limit was increased to 720, it would spin around twice. In addition to the initial spin, the spin appears to speed up until 180 degrees then slow down to make the start and stop of the spin less abrupt, which was achieved by speeding up the spin rotation until 180, then slowing down between 180 and 360 for a nice slow stop to the spin. All in all this amounts to a pretty cool effect, that wasn’t difficult to add.

**Challenges**

One of our biggest problems was getting the cube to appear on top of the track, for quite a while, the cube was not being rendered properly, and just showed up hidden below the track, and at the wrong angle. It was known what needed to be done, but actually getting it to happen correctly was causing quite a bit of a problem. We tried many different ways of getting it to appear properly, but all attempts ended up giving weird, incorrect, confusing appearances of the track or player model, or the combination of the two. To finally fix this we ended up using the same mvMatrix that we did for the track to render the cube, but then applied a scaling factor to shrink it down to size, and then translating it to the proper position in the world using the playerPos variable, and a constant Y adjustment to make it appear above the track, instead of inside it. Once this was applied, everything started to render perfectly as desired, the scene looked correct, and then the game worked as intended and was playable. Other problems were insignificant.

**Notes**

Progress in the beginning was slow, we initially started the assignment with different partners and then were paired together when the others fell through.

# **Screen Shots**

### Start State:

# 

### Losing Screen:

# 

### Winning Screen:

# 

# Status Report (11/16)

Our project is designing a game in which the user controls an object on a platform high in the scene. The player will navigate the object through a course through the sky. If the player should drive off the course, they lose.

Project Status Report:

Being completely honest, we haven’t made much significant progress toward completion of our project. As of now, we have it so that the player spawns in the center of the track, and then navigates the track by using the mouse to go forward, backward, and steer. The track being built from a height map is working, functionally speaking, but it looks horrible, so this will need some attention. The player still is just a camera, so addition of an object model in third person view is still required. Currently, we have a simple lose condition working, so that if the player drives off the course, there is an alert given that the player lost, and their position is reset to the start of the track. There is a lot of work to still be done. The majority of what is done is simply what was used to complete assignment 3, but at least it’s a start. The project breakdown is as follows:

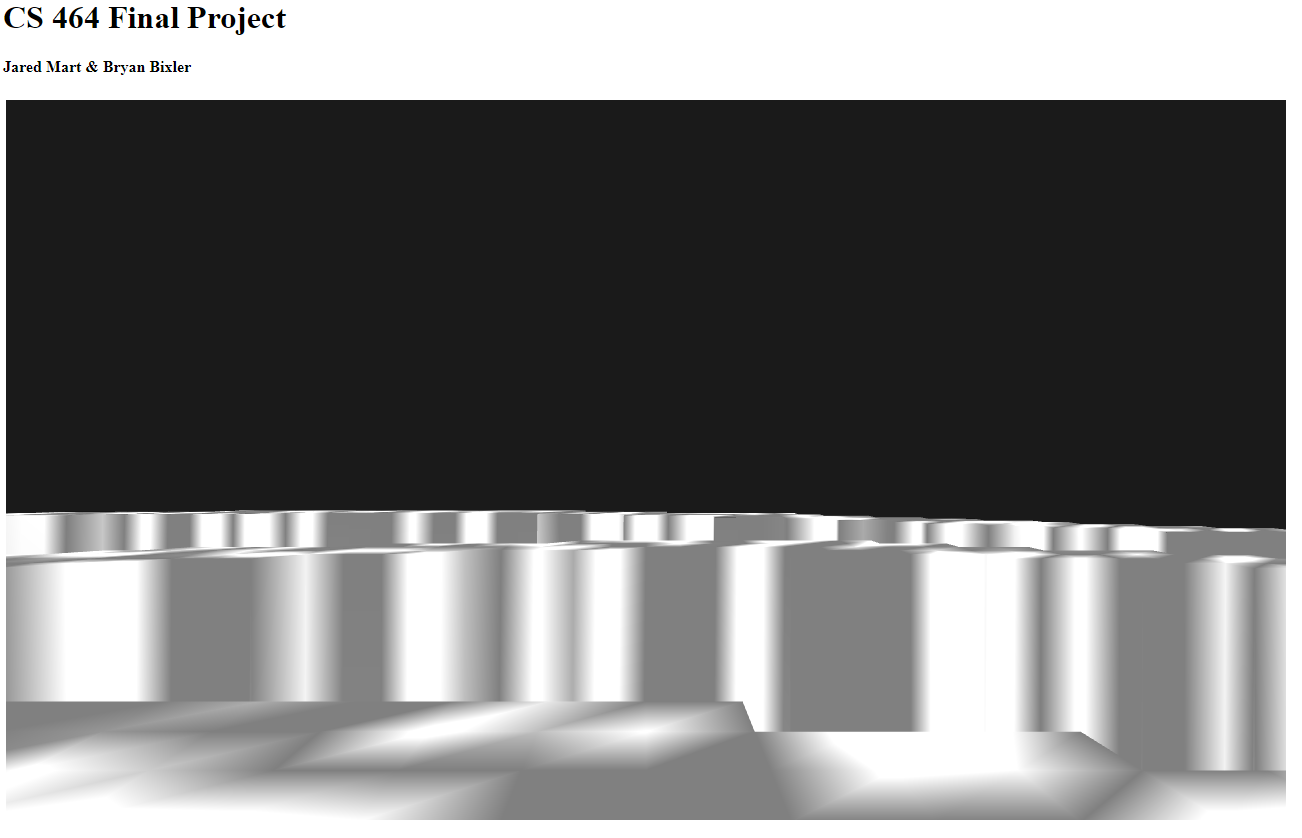
What is working:

* Player Navigation
* Generation of a track from a texture
* Lose condition

What still needs done:

* 3rd person viewpoint with addition of player object model
* Better looking track with no nonvalid track displayed
* Physics (mainly gravity, but navigation also would be preferred)
* Win condition

Screenshot of current status on next page



# Project Proposal (10/26)

### **Brief Description of Concept:**

For the final project this semester I plan to build a relatively simple 3D game. This game will be a 3D platformer, in the form of a character model (I haven’t decided whether it will be a ball, car, etc.) trying to slowly navigate an airborne obstacle course. The intent is that the course will feature a sort of tight-rope course that the player needs to successfully navigate. The player will start on a beginning platform and then attempt to navigate the course by moving and turning their character model over the course. The player will complete the course once they safely reach the ending platform. If the player makes a mistake and moves their character over open air, the model will fall off the course and into the abyss, and the player loses. The game will be level based, so there will be an initial level, and once the user completes that level it will automatically move on to the next level.

### 

### 

### **Implementation Details:**

I would essentially like the levels to be generated by the game from a texture image. I plan to do this using a binary representation of the course with the pixels of the image. The way this would work is a level starts out as an all while image, then to create a level, all one would have to do is draw a black line in the image file to represent the track. When the game then loads the “level” from the image texture, it would essentially create a height map of the image in the scene, with the white being open space, and the black being the track, so the black part of the image would be high in the scene, and the white would be at the lowest. Thus when “gravity” is simulated on the character model, if the player moves off the black part of the texture (the track), then the model will fall off, and if the model falls off the track, they lose. This could be done by firing a lose event if the model’s y value falls below a certain value.

I’m not sure if it is allowed, but I would very much like to use an existing physics engine inside of my webGL code to simulate the physics in the game (gravity, driving forward, etc). I could probably get it to work on my own given enough time, but creating a physics engine is a bit out of the scope of this project in my opinion. The class is about graphics, not simulating physics with mathematical code, so I hope this is okay.

The movement of the character model would be done by translating the geometry of the character model over the scene.

The camera would follow the character model in third person view from a small distance away.

The character will be controlled with arrow keys or mouse drags on screen. I haven’t yet decided which. I may do both if time permits.

There will be simple lighting in the scene, likely goraud, maybe phong if I have time.

If all is easily implemented, I would want to add more UI elements and game enhancements, such as maybe a timer to beat level high scores, a minimum time to beat the level, a menu, etc.