3D Mini Golf Game

Ari Riggins, Danny Sun, and Nasko Tenev
Professor Felix Heide
Advisor: Joanna Kuo

This project implements a 3D Mini Golf Game where players can control the golf ball with their keys in order to get it in the hole and advance to more complicated and fun levels. Our implementation features THREE.js meshes, Scene and Animation Rendering, and a lot of physics to help with collisions, ball movement, and hole success indication. The game uses event listeners for pressing certain keys from the keyboard in order to adjust the ball's direction and launch power the ball. The game is designed to go through the different levels on your own, using the keyboard to navigate and shoot the ball, get it in the hole, and advance to the next course.

1. Introduction

Goals: Our project aims to create a 3D Mini-Golf Game with multiple levels and a way to navigate the ball to the level's hole. We created different scenes with THREE.js meshes and created a ball and hole object to be able to create different mini-golf courses. Additionally, we created a transition method to change the different scenes once the ball goes in the course hole and settles. Our goal was to create three different courses and a smooth way to navigate the ball and transition between levels/courses.

Inspiration: Our inspiration for the project came from looking at other mini-golf games online. Some example games that we looked at were mainly from Google images just to try to find a good sample and inspiration for the project. We found inspiration from games such as Mini Golf 3D Multiplayer Rival on Google Apps[1] and the PS game 3D Mini Golf[2].

We didn't find any specific examples of previous projects in THREE.js that did something quite similar to our approach so we just used a lot of primary elements such as meshes and skills from other assignments, such as Assignment 5: Animation, in order to create the golf courses and make a well-transitioning game with several levels.

Approach: We implemented our game by using THREE.js meshes and adding many physics components to the objects we created. In essence, we basically used the Final Project starter code given to us and added more objects, scenes, event listeners, and HTML div components to create the game design and user interface. We created the course, the ball, and the other objects in the game by creating meshes and making collisions for them. In those specific courses and object meshes, we added many different functions to handle updating ball position, velocity, and acceleration as well as created a lot of state variables to indicate whether the ball is moving, it's in the hole, and when to transition. We also used event listeners from the keyboard in order to see

[1] https://play.google.com/store/apps/details?id=com.MobileSportsTime.MiniGolfEldoradoGolfCourse&hl=en_US&gl=US
[2] https://www.amazon.com/3D-Mini-Golf-PlayStation-4/dp/B0778W3S11

if the right keys adjust ball direction, power, and shoot the ball for the player to interact with the game. We also had many different embedded HTML components into the document that appear and disappear based on certain conditions in order to display game logistics, and transition screens between levels.

2. Methodology

As with many large coding projects, starting the code and getting your bearings is probably the hardest part. This is especially true when you have a general image in mind but have no concept of how difficult certain jobs are or even how to split your final idea into discrete steps. When we began with the starting code, we were given a floating island with a happy little flower, we also had many reference projects from previous years. However, while the code gave us everything we needed to see how the renderer, camera, and scene all worked together, it was only much later on that we actually fully understood how they worked individually.

Instead, we started off by trying to create some sort of floor and some sort of ball that would be the building blocks of everything that came after it. We rendered the scenes using the Final Project and started to code with the render loop in app.js, which just tracks the changes using a timeStamp component. This allowed us to initially display the course scene we developed. It seemed like a good idea to simply change the more specialized files, i.e. the flower and island objects since the scene and renderer files already worked to create something. Plus, it was easy to see that the scene simply called a constructor to create some sort of mesh and all we had to do was create new constructors. We sought out guidance from assignment 5 where we had already done some sort of physics interactions and animation and after looking through that code, we were able to make a mesh of a plane and a sphere with some simple pre-built functions from THREE.

[Golf] Ball Position, Velocity, and Acceleration Update:

It was not much harder to then make it so the ball would have some physics to update its position as well as some collision detection so as to not fall through the floor. First, we had to understand how the update() function was used in the scene.js file. Then, we began adding the physics by keeping instance variables for the ball's position, velocity, and net force vectors similar to that in assignment 5, but instead of using the Verlet integration formula, we used the method from COS126's n-body which would update each of the three instance variables at every timestep. We thought this method would be better since it would explicitly keep track of the velocity which we knew would be useful later to control the ball's launch. It is important to note that while the update function has a "timestamp" parameter, it keeps a running clock of milliseconds and can not be used as a step. Instead, looking at assignment 5, we used their step value of 18/1000.

Initially, we only had a force of gravity acting on the ball, but later added a drag, or friction, a force so that the ball would eventually stop moving on the floor. The basic idea is that during each update, in addition to gravity pointing in the negative y-direction, we had a drag that pointed opposite to the direction of velocity. Just as in many real-life physics interpretations of drag, the force would be proportional to the velocity (we tested various values for the constant of proportionality and just eyeballed which value looked realistic). All of these forces and updates can be found in the Ball.js file under the objects folder of the repository.

Box Construction:

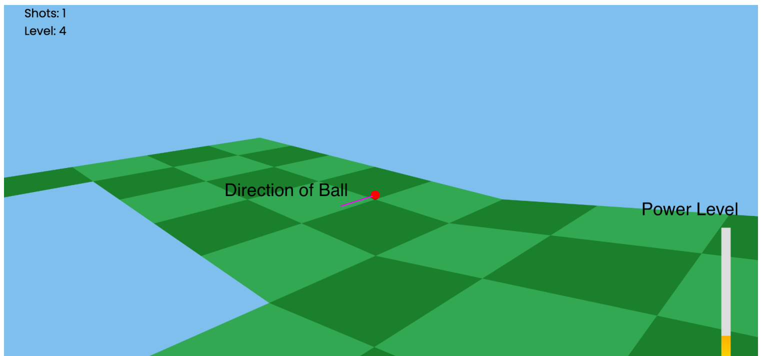
We were unable to use the three js premade box in our project, so we constructed our own box meshes out of collections of planes as seen in our level 2. By using location shifting and rotating, our box constructor builds a 2x2 box object centered at the input x,y,z location. It is constructed with a bounding box property which is used to calculate the balls collisions with the box.

Floor Collisions:

It was then necessary to handle the collisions with the floor. We began simple and moved our way up in complexity as we wanted to add more degrees of freedom to our plane. First, we began simply with a plane parallel to xz at y = 0 and followed the work from assignment 5 and checked to see if the ball was below it (with some offset of the sphere's radius and EPS value) and would push it back above it if so. However, we also wanted to add some bounce if the ball dropped so that it would look more realistic. Basically, we would flip the velocity vector and lower it by some tested constant if the velocity was above some magnitude (this avoided the issue of infinite bounces, if the velocity was small we set it to zero to stop the bounce). The physics for this section can be found in the Ground.js file under the objects folder of the repository.

Floor Bounds:

Later on, we realized that we might want to add slopes and have multiple planes and limit the size of the plane. We began by adding these as free parameters to control the plane's location, rotation, and size, and then did some math to find where the corners of the plane would be projected onto the parallel plane by getting the original coordinates, rotating them, and shifting them (it was interesting to see that the rotation axis also rotated with each rotation). Then, by splitting the plane into two triangles, we could test if the ball was true atop the plane by using the built-in containsPoint() function and skip the collision testing if the ball was not atop the plane. Finally, instead of just having the floor at y=0, we had to use the equation of the plane to find where the proper y value would be given the x and z coordinates of the ball if there was indeed a slope. The physics for this section can be found in the Ground.js file under the objects folder of the repository.



Aim/Shot Mechanics:

We then moved on to creating some way to aim and launch the ball and started off by looking at the listener used in the Portal 0.5 project by Allen Dai and Edward Yang[3]. It was a simple key listener that would keep track of when a key was pressed, held, and released. We tried multiple control schemes for launching the ball but ended up landing on changing the direction of launch with the a and d keys along with a hold/release of the space bar to launch with a specific power. We kept an instance variable for the direction of launch and then created a line (later a tube for more thickness) that would point from the ball in that direction. When the spacebar was released, it would give the ball a jolt of velocity in that given direction.

One thing we wanted to make sure of is that we wanted the ball to only be able to launch when it has been set to rest. To do so, there were instance variables that kept track of whether the ball was moving in the x-z direction and if it was falling. Once these conditions were met, we would set a variable to the timestamp (otherwise it would be null), and then only after these conditions were met for a certain amount of time (subtract current time with saved first time) would we give the player the ability to aim and shoot. There were a couple of other nitpicky things that we ironed out such as projecting the aim along the plane it was on, how to display the powerbar (basically change the width of the CSS), and what the power was (we used a sin function). The shot mechanism can be found Ball.js file under the objects folder of the repository.

More Out of Bounds Details:

There was a point later on in the creation where we realized that we had to recover the ball if it were to fall off the edge. To do so, each time the ball came to rest we would save that position and return the ball there if it were to go out of bounds. In this case, we just set out of bounds to some negative y value that it was not allowed to go beneath.

Box Collisions:

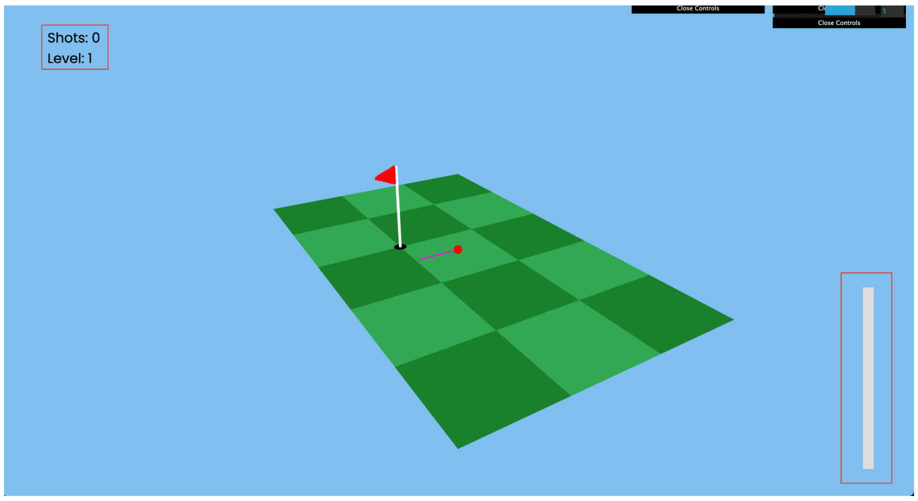
It was also necessary to handle the collisions with the box obstacles we created so that the ball would not roll right through them. In order to implement this in the box constructor we create and save a bounding box with the minimum and maximum coordinates of the box planes which is then accessed within the ball method handleBoxCollison. Here we check if the x and z coordinates of the ball fall within the bounding box and if they do we apply a reflection vector to the balls velocity such that it bounces off of the box. We also found it necessary to check which wall of the box the ball contacts in order to determine the proper vector.

Hole Mechanics:

The last important thing was the creation of the hole for the player to launch the ball into. Instead of creating a variable on its own, we thought it would be smart to attach the hole object to a ground object so we could slightly simplify the geometry and collisions. Basically, the hole was a black circle (later we added the flag with the tube geometry) that sat above the plane at some specified point and we would test to see whether the ball's position went on top of the hole.

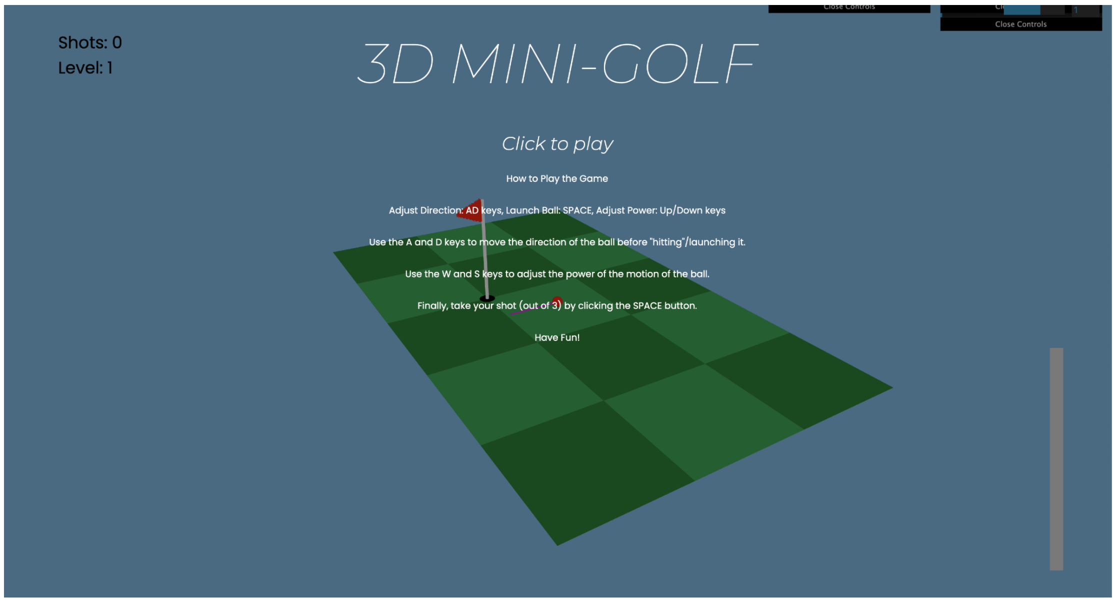
[3] https://github.com/efyang/portal-0.5

If so, we would change the floor collision such that the floor's y value was lower by a value of 0.2 allowing the ball to drop down into the floor. We also had to make sure that the ball wouldn't escape the hole so I added collisions inside the hole so that it was basically a cylinder. Each time the ball would hit a wall its velocity would be reflected using the normal from the center of the hole to the point on the wall. There was then some testing for whether the ball was in the hole.



Displaying HTML Elements:

In order to make the game smooth and understandable, we added a couple of HTML elements in order to display the number of shots a player has made, the current level/course of the game, and a power bar for the power level of the ball that is based on a sin curve fluctuation. We embedded these elements into the game by creating 'div' elements in the respective CourseScene.js files with the text elements in the innerHTML and appending those elements into the document body. For the power bar, a similar implementation is done but with the innerHTML being a separate HTML file.



Shots: 0
Level: 1

3D MINI-GOLF

Click to play

How to Play the Game

Adjust Direction: AD keys, Launch Ball: SPACE, Adjust Power: Up/Down keys

Use the A and D keys to move the direction of the ball before "hitting"/launching it.

Use the W and S keys to adjust the power of the motion of the ball.

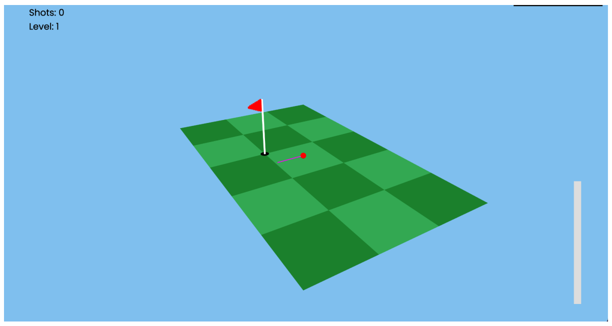
Finally, take your shot (out of 3) by clicking the SPACE button.

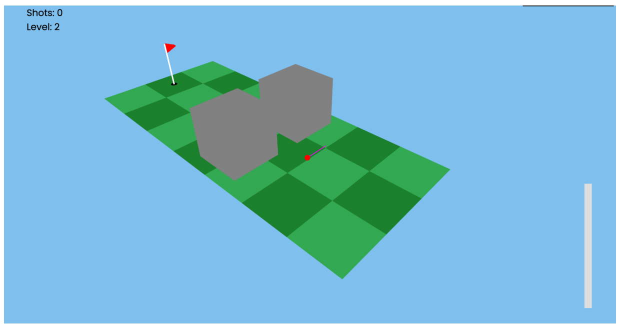
Have Fun!

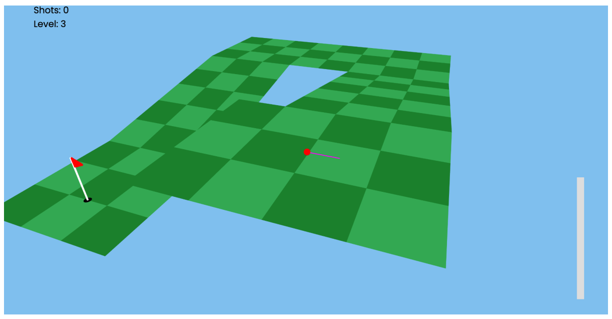
Transitioning Between Scenes/Levels:

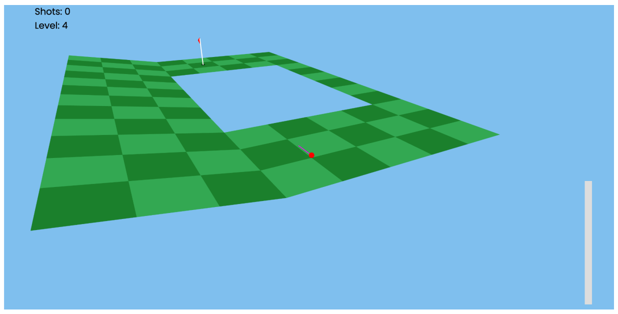
To transition between levels/courses, I made the scene variable in our renderer in app.js an array of the different course scenes we have created and I created a level counter to switch between levels. The counter increments when the ball is in the hole, which is tracked within the render loop with a function passed by each scene getBallSuccess(). Once that is true, meaning the golf ball went inside of the hole, then the level counter increases, and the render loop loads the next course unless it is the last level. Additionally, it loads new HTML div components, defined as HTML files to project the beginning screen of each level, including the intro screen. This allows for smoother gameplay and transition between courses.

3. Results









Our resulting game consists of four different levels. The first level/course just aims to have players understand the controls of the game and displays a simple course. The second level introduces obstacles and courses 3 and 4 have both obstacles and tricky inclines for the player to navigate. The game is posted on GitHub Pages and can be played by a single player. To calculate your score, you simply aim to complete the level in the least amount of shots possible. The game would be amazing to play with two different screens and devices and have players play together to compete against each other.

We measured success by going through the game and making sure the collisions, falling off the ball, and the transition between levels is successful. We were able to complete the entire game without any errors and the transition between levels is smooth and seamless for players. Additionally, there is no lag for the game unless the device itself is overwhelmed with other programs running in the background.

Our gameplay was really smooth and we believe we implemented a successful game. We hope others get to enjoy the game and play with other people at the moment before possibly developing multi-player games.

4. Discussion

The approach we took did work, but it involved a lot of debugging, referencing other projects from past years, and ultimately adopting an approach that best fits us. We had to pivot a lot or find creative ways to make the shapes and boxes work for our game. For instance, it was hard to implement the box obstacles with the collisions so we had to create a lot of collision adjustments.

We honestly did not have time to figure out a different approach to this project because we just worked a lot to make the current approach work and we didn't have much time to make realizations about what would be the best approach. A lot of extreme programming took place.

By doing this project we learned that there is so much potential for it and it was hard to see a stopping point since we could always develop it and make it look better, especially with only examples from the top projects in previous years. We think the timing and the example projects to go off of gave us a lot of doubt about the looks of our project but we are ultimately proud we developed a project with a lot of moving components and detailed physics.

5. Conclusion

We think we implemented a really cool golf game that has a lot of potential for some really cool levels and developed a lot of collisions and physics components and make for a very realistic golf experience. We think we implement this project quite successfully, even though we did not have time to possibly polish or add all of the details we wanted.

For future steps, we plan to add some audio to the game, similar to how other projects have done it in the past, we also plan to make more barriers, and or levels if we decide to continue developing the game. Additionally, we hope to add some backgrounds and environments for the game to seem more realistic to play.

6. Contributions

- Ari: Developed the second course of the game and a Box Mesh with collisions for the second level and beyond. Also started working on background details such as adding water or a sunset or other landscapes.
- Danny: Developed the Ball, First Course, Hole, Flag, and Physics of the assignment. Developed a Power Bar and worked on adjusting the collisions of the floor to other courses. Also Developed Courses 3 and 4.
- Nasko: Developed the gameplay transitions of the game. From the beginning, the screen to the level changes the screen and displays the statistics of each level. This also involved helping to transition between different scenes (A.K.A. golf courses).

Works Cited

https://www.amazon.com/3D-Mini-Golf-PlayStation-4/dp/B0778W3S11
https://play.google.com/store/apps/details?id=com.MobileSportsTime.MiniGolfEldoradoGolfCourse&hl=en_US&gl=US
https://github.com/efyang/portal-0.5
https://threejs.org/docs/
https://www.w3schools.com/howto/howto_css_skill_bar.asp
https://stackoverflow.com/questions/30245990/how-to-merge-two-geometries-or-meshes-using-three-js-r71