Lawnmower Lunacy

Henry Knoll, Silas Mohr, Brian Tieu, Brett Zeligson

Abstract

Welcome to **Lawnmower Lunacy**! Our project is the ultimate lawnmower game, where the goal is to maximize your lawn mowing score by cutting as much grass as possible, while avoiding rocks, in under a minute. In completing this project, we accomplished our minimum viable product of creating a game where user controls could determine the direction and speed of a lawnmower cutting grass. Extending this MVP, we added rock obstacles to the lawn, a pleasing backdrop of a farm, tall weeds that result in lawnmower size boosts, options for changing difficulty and detail, grass that sways in the wind, and optimizations to our data structures and algorithms. We believe we had a successful outcome and are super happy with the final product we created.

Introduction

As Princeton students, we miss the relaxing afternoons in the sun spent mowing our backyards, smelling the fresh cut grass, and listening to country music—so we decided to make a game for it! Lawnmower Lunacy is the quintessential lawnmower game for those of us craving the satisfaction of mowing a neatly-cut lawn. The purpose is simple: cut as much of the lawn as you can in one minute, while maximizing the points you collect by avoiding rocks and cutting tall grass for a size boost. You'll feel right at home, as if you were in your own backyard!

Goal

Our goal was to create a lawnmower game that took in user input for lawnmower speed, direction, and mechanisms for cutting grass. After reaching this goal, we moved into implementing additional features including texture mapping, an on-screen control panel, collision detection, sound, vertex/fragment shaders, simulated dynamics, a countdown, level of detail control, and multiple views, as well as other features to enhance gameplay and the user experience.

The target audience for our project encompasses gamers of all ages and skill-levels. The simple interface, controls, and lawn mowing idea were all purposefully designed to make the game accessible, so even those who have never picked up a game before can enjoy the serene calmness that comes with mowing virtual grass. Additionally, the

use of semi-realistic graphics of the outdoors and soothing music functions to make the game a relaxing and engaging experience for all.

Previous Work

Most examples of lawnmower games are either overly complex, meant to be used as realistic simulators of lawnmowers in the real world, or very simple, with uncomplicated graphics, collisions, and user interfaces. Thus, we aim to fill this gap by creating a game that has more complicated features such as collisions with multiple objects, a user interface, and semi-realistic graphics.

Approach

In designing our approach for the project, our first step involved determining the components that would define our MVP. We defined these components to be a three-dimensional lawn terrain and lawnmower with functionality to cut grass as the user steers the lawnmower with the use of arrow keys. Once these steps were completed, we implemented the rest of features described in this document that satisfied the feature requirements of the project and further enhanced gameplay and user experience. Also, it is important to note that throughout the development process we split the work between us such that one member focused on the lawn, another on the lawn mower, another on the obstacles, and the last on game design.

Additional Features Implemented

1. Texture Mapping
2. Multiple Views
3. On-Screen Control Panel
4. Vertex/Fragment Shaders
5. Sounds
6. Collision Detection
7. Simulated Dynamics
8. Level of Detail Control

Methodology

For the **On-Screen Control Panel**, which consisted of the timer, score, screen view toggle, start page, and end page, we found that the most practical implementation would be to add a element via the app.js file and then use the innerHTML property to write custom HTML that would be displayed. Additionally, in order to add custom styling, inline CSS was utilized. The advantage of using this implementation

involved being able to build out an appealing interface However, one disadvantage involved the HTML and CSS being very verbose, leading to us using more clever logic later in the game in order to properly start off the game. Additionally, one issue we encountered involved rendering the scene while taking into account the user input for difficulty level and level of detail. Originally, we rendered the scene when the site loaded, but we ultimately had to delay scene rendering until after input was attained, so that the proper difficulty and level of detail were present in the game.

For the grass in the game, we started off with a basic implementation from JSfiddle linked here. This JSfiddle includes a patch of grass as a single Instanced Mesh where each blade of grass is in a random position between the bounds of the patch. Originally, we created our lawn with multiple Instanced Meshes corresponding to different subpatches of grass. In this implementation, whenever the lawnmower went over one of these subpatches, all of the grass in the subpatch would get removed. This theoretically made sense if the subpatches were small enough, but didn't look great in practice. We wanted to find a way to calculate precise **Collision Detection** where individual blades would be accounted for, instead of patches. To do this, we made each blade its own Instanced Mesh. To make sure that the grass still looked like a lawn, we divided our lawn into a grid where each grid box was a section where an individual grass blade would be placed inside randomly. The sizes of these grid subsections are proportional to the level of detail control input. Larger sizes sacrifice density of grass for faster rendering. We used spatial hashing to keep track of where each of the blades were. The XZ plane of grass was broken up into spaces corresponding to numbered indices, so that with O(1) lookup time we could find the blades within a given space. Therefore, spatial hashing allowed us to efficiently find which blades of grass needed to be removed according to the center of the lawnmower and a radius surrounding it (like a spinning blade on a real lawnmower). This avoided iterating over each blade of grass, which would've been massively inefficient, resulting in serious lag.

In order to **Simulate Dynamics** of grass swaying in the wind, we created a **Vertex/Fragment Shader** that adds a subtle movement to the tip of each blade of grass according to a sine wave paired with random noise. The random noise in the shader is pseudorandom and relies on modulus operations according to the position of each blade to ensure that all blades of grass have slightly different movements, which makes the effect of a realistic breeze.

The lawnmower is a static model that we modified from Sketchfab. It includes a state struct that holds the current velocity, the maximum velocity, which direction is forward and the radius that you can cut with. The lawnmower Object3d also contains two child objects, a basic Object3d that the camera uses to position itself with respect to the

lawnmower and a bounding box that is just an invisible box mesh. These two objects keep their local position with respect to the lawnmower, but move around the world with the lawnmower. There is also the audio file that contains the lawnmower's **Sound**, which plays with a volume proportional to the velocity of the lawnmower. The movement is controlled by an event listener that listens for key presses, applying either rotation or acceleration in velocity. On update, the lawnmower moves along the forward vector proportional to velocity and calls applicable collision functions.

We also needed to place interactive elements on the lawn. We placed rocks that would stop the lawnmower if a **Collision** occurred and patches of tall grass that, when mowed, offered a temporary power-up of an enlarged mower with a greater mowing radius. To implement the rocks, we generated three spheres of random size, offset against each other. A random offset was applied to each vertex to make the spheres appear rockier, and each sphere's radius and center point is stored in an array. If the bounding box of the lawn mower comes too close to a rock's center point, it is projected outward to the outermost edge of the rock, and its velocity is set to zero. Another approach we considered was to import existing rock models and randomly disperse them; however, this would not allow for the granular level of detail control we were able to implement, and allowed for much less variability in the random generation of rock shapes.

For the trees in the game, several possible implementations existed. The first implementation involved a simple geometry consisting of a cylinder and three cones stacked into a tree-like shape. Then, we used an InstancedMesh in order to place these trees around the lawn in random positions with random scalings. The advantage of this implementation was that it was fairly efficient, but the disadvantage was that the trees didn't look realistic. For this reason, we chose to find a more complex 3D model of a tree online to replace the old model with. One issue we ran into when implementing this new tree involved not being able to use an InstancedMesh to place trees as we did previously. The solution to this issue involved loading the GLTF file and then adding elements to the gltf.scene.children array to add more trees. With each addition to the array, the position of the newly added tree as well as its scale factor were randomly chosen. Additionally, since these trees are more complex than our previous iteration, a bit of efficiency was sacrificed, but we thought that it was worth it for game aesthetics.

The tall grass is generated in a similar fashion to the normal grass; however, there is an extra condition that checks if the grass blade is within a certain radius to a randomly generated set of ten points on the grid, which allows it to grow in ten spherical patches. Again, if the lawn mower enters one of these patches, its scale is increased and the point is removed from the set of points, to avoid further collision detections and an infinite increase in size.

The fence is a static model that was created by Silas and is textured with a **Texture Mapping** found on Textures.com. The **Collisions** with the fence are done by checking where the position of the lawnmower is. If it is outside of the lawn (i.e. the area that is fenced off), the fence pushes the lawnmower back inside the lawn area.

The farm in the scene involved downloading a 3D **Textured** model of a farm online and then providing some scaling, rotation, and position changes to properly orient it in the scene. We felt that the farm was an important aspect of the game because it provided not only nice scenery but also a bit of a storyline, convincing the user that they were working on a farm. Additionally, a grass **Texture Mapping** was applied to the ground by loading an image of grass, setting that image's repeat property to repeat both horizontally and vertically, and then mapping the texture to the plane geometry.

The camera has **Multiple Views**, a chase camera that follows behind the lawnmower and a fixed offset from the lawnmower that always faces towards the barn, but follows the lawnmower. The chase camera was implemented by creating an object within the lawnmower that followed the lawnmower and rotated the same amount, so that we always had a goal position for the camera. The position of the camera was calculated by interpolating between the last camera position and the goal position of the goal object. For the second view, the camera's position was calculated by simply finding the position of the lawnmower and adding a fixed offset so that the camera always followed the lawnmower.

An important measure of success for us was the game loading speed and the frame rate of the game. In order to accommodate different types of computers and users, we implemented a **Level of Detail** control that varied the granularity of the rock design and the number of grass blades rendered. Lower levels result in much faster rendering, whereas higher levels result in enhanced rendering, potentially introducing lag.

Results

We ran the game on multiple devices to ensure that it functioned well on a wide range of processing units. Our results indicate that a wide variety of machines are able to play our game at a sufficiently high frame rate, provided they choose the correct Level of Detail setting for their machine.

Another way we measured success is the level of engagement and fun had by players. We measured this by allowing several test users to play our game, and offer us verbal feedback. We received highly positive feedback, with many users praising our graphics,

high quality of collisions, and engaging gameplay mechanics. To this end, we believe our game is successful.

Discussion

We feel that the approach we took was very productive. We divided tasks well and rarely ran into issues when we were working concurrently. Furthermore, through completing this project, we not only learned the complexities of ThreeJS and 3D game development, but also the best ways to prioritize and delegate tasks to complete a large-scale project in a short time frame.

Conclusion

Looking back on our finished project, we feel that we achieved our goal completely, creating an engaging, well-designed lawnmower game that possesses all the functionalities of our planned MVP and our additional features. In the case that we continue developing this project, our next steps would be to add dynamic obstacles, for example squirrels running around the lawn, to make the game more interesting. Another next step could be to further optimize the code base such that there is no lag for higher detail renderings. Currently, there are no issues that we know of in the code that need to be revisited in the future. In the long-term future, this could be a great opportunity for a virtual reality game!

Contributions

Brett: My contributions to the project included all of the user interface design, the game design (including the countdown timer, scoring, toggle button for camera view, and play again functionality), as well as implementing the farm, the trees, difficulty for the user (adjusting the number of rocks based on user input), and all the texture mapping for the scene aside from that which appears on the fence.

Brian: I focused on the generation, placement, and collisions with objects generated in the scene, including the rocks and the tall grass. I implemented the random generation of various rock geometries as well as their collision hitboxes, patches of tall grass and the handling of scaling the lawnmower and its mowing radius upon collision, and sound effects for each of these. I also implemented a Level of Detail control, which affects the number of polygons for objects within the lawn.

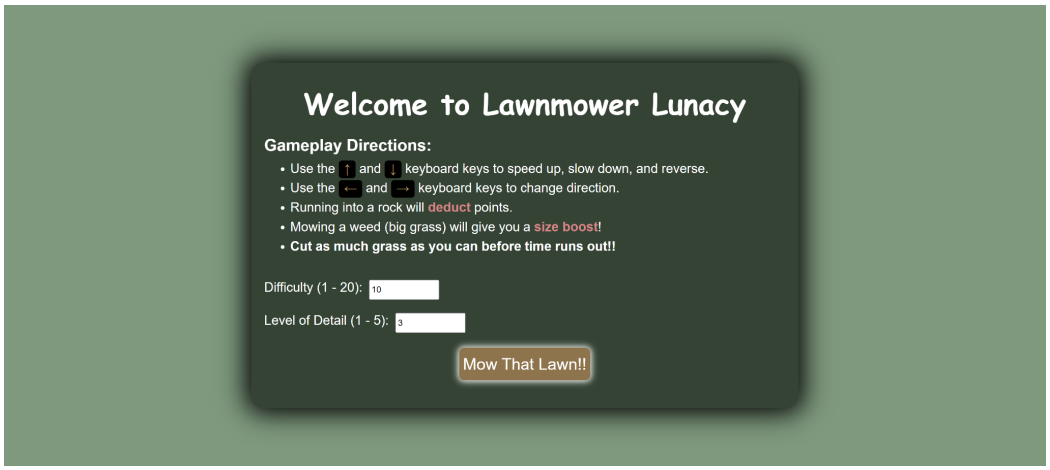
Silas: I focused on the lawnmower, camera, and the fences. I implemented the lawnmower first, adding the acceleration and turning, as well as finding and editing the

final model. I also found the correct dimensions for and added the bounding box for rock collisions. I then implemented the two camera views and made sure that they correctly followed the lawnmower. I also modeled and textured the fence in Blender, and then implemented collisions with the fence.

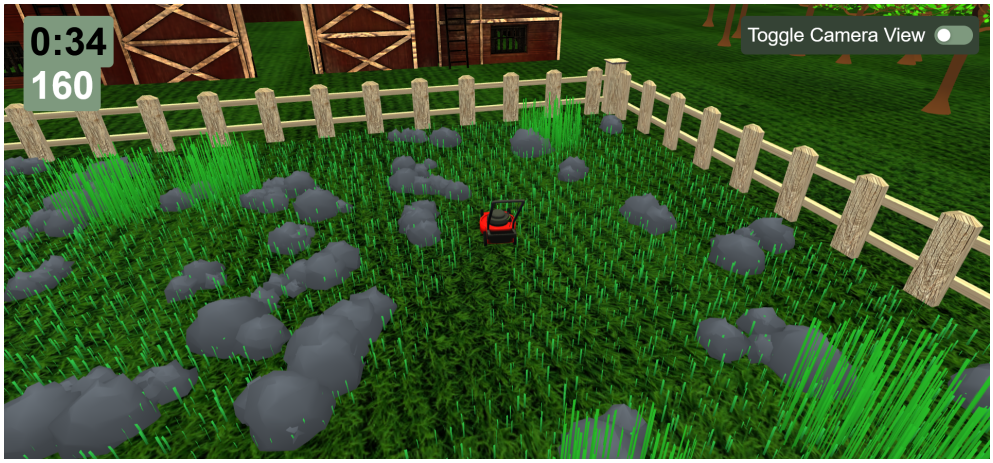
Henry: I focused on building out the grass, whether that was implementing the spatial hashing for collision detection, the swaying of the grass via the vertex shaders, different generation levels for different detail inputs, or the disappearance of grass after being cut. I also worked on making sure the audio files worked well, including looping and matching. This was an especially rewarding project for me, and I thoroughly enjoyed it.

Example Screenshots

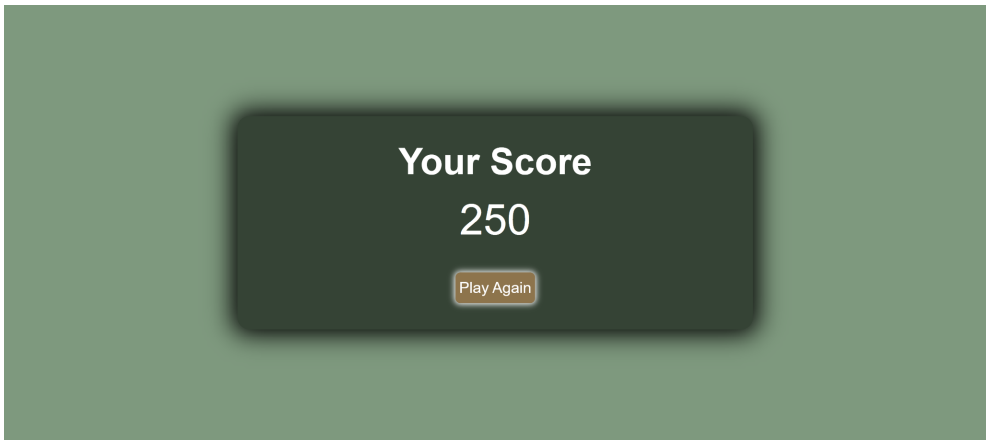
Start Page



Gameplay



Endpage



Works Cited

Technologies

- HTML
- CSS
- JavaScript
    - ThreeJS

External Resources

- Codepen.io
- Stack Overflow
- Blender
- ThreeJS Documentation
- Sketchfab.com
    - Some models were created by us
- Textures.com
- Flaticon.com

Visual Assets

- Ground Texture
    - Aerial view of rice, agricultural fields in countryside of Taiwan in spring season. Rural area. Farm pattern natural texture background. Stock Photo | Adobe Stock
- Barn

    - https://sketchfab.com/3d-models/farm-low-poly-116065ff36c64fe9b8e8edcf6c382a50
- Trees
    - https://sketchfab.com/3d-models/stylized-tree-6d1aeea748f147789004bc03e1930d32
- Lawnmower
    - https://sketchfab.com/3d-models/lawn-mower-low-poly-be7ab00cce174ef1b6045017493591c6
- Favicon
    - https://www.google.com/url?sa=i&url=https%3A%2F%2Fwww.flaticon.com%2Ffree-icon%2Flawn-mower_3056971&psig=AOvVaw38Ux9280Bf0gA9G40hdZpv&ust=1671315536697000&source=images&cd=vfe&ved=0CBAQjhxqFwoTCNiu-duV__sCFQAAAAAdAAAAABAD

Audio Assets

- Rock Collision 1
    - https://www.zapsplat.com/music/large-stone-drop-on-forest-floor-with-a-thump-1/
- Rock Collision 2
    - https://www.zapsplat.com/music/large-stone-drop-on-forest-floor-with-a-thump-2/
- Tall Grass Collision
    - https://www.zapsplat.com/music/retro-arcade-game-level-up-1-up-win-4/
- Country Music
    - https://pixabay.com/music/solo-guitar-the-beat-of-nature-122841/
- Countdown Beeps
    - https://pixabay.com/sound-effects/game-start-6104/
    - https://pixabay.com/sound-effects/start-13691/

Inspiration Credit

- Inspiration for Rock Generation
    - https://blog.mozvr.com/procedural-geometry-low-poly-clouds/
- Inspiration for Camera Movement
    - https://jsfiddle.net/Fyrestar/6519yedL/
- Timer
    - https://codepen.io/ishanbakshi/pen/pgzNMv?editors=0110
- Grass
    - https://jsfiddle.net/felixmariotto/hvrg721n/
- Inspiration for Grass Swaying
    - https://codepen.io/al-ro/pen/GRJzYQK