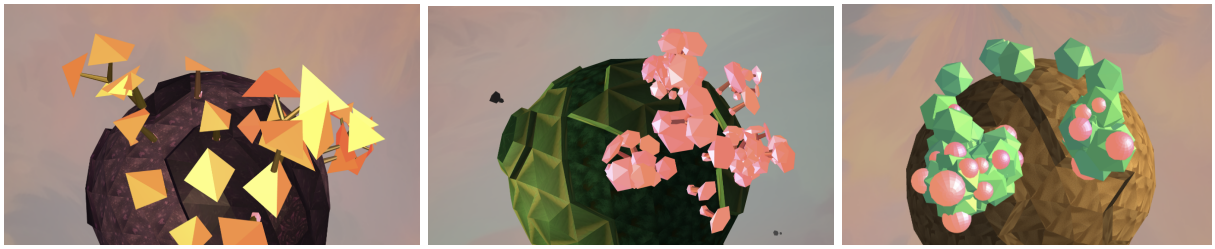
CPSC 478 Final Project: CEEDER (https://dsmaugy.github.io/ceeder/)



(Figure 1: Three images of three different planets and their unique tree that can be planted on them)

A created by Jack Li, Raff Davila, Darwin Do, and Samantha Trimboli

Abstract

With the semester coming to an end; final papers are being turned in, final exams are being taken, and final projects are being submitted, how can we rest? Wind down before in between and after the last academic hurdle? Our team presents, CEEDER; a space-themed simulator. The player can relax while planting unique trees on three different planets. They can continue to grow each of the plants between 1 and 7 stages by clicking on each of them.

1. Introduction

Our goal was to create an interactive space zen garden game. The player would have the ability to populate their chosen planet with seeds and watch them grow over time.

The game is meant to offer a relaxing world-building experience for users. While other games require time-consuming energy or the emotional stress of winning losing and completing levels, our game offers a break from the competition. Our target audience would be overstimulated college students that need a 5-10 minute break from their work.

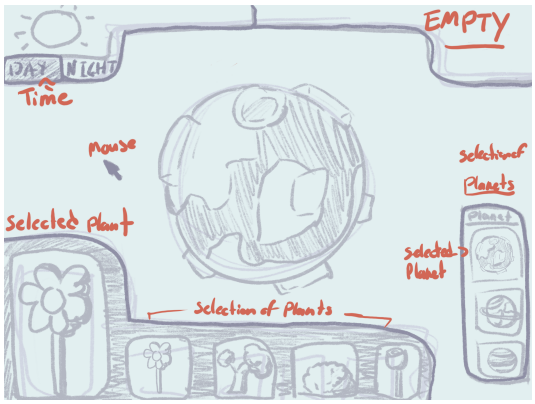
The primary mechanic of the game takes from farming simulators. In games like *Farm Story*–a popular mobile game released in 2010– or Stardew Valley, the player has a certain amount of seeds they can plant, grow, and harvest to make profits/expand their farm. A game like this creates a reward system based on the farming mechanic. Although was want to take the planting and growing, we don't want the player to harvest. The planting aspect is supposed to serve a purely aesthetic purpose. Similar to the build feature in the Sims franchise or creative mode in Minecraft.

1.1 Initial Goal

We initially decided to divide the work into three major categories: Animation, UI design, and Rendering. The animation aspect would be how we the plants grow overtime. The constructed UI would help the user change between the planets and plants they want to work with. Finally the rendering would how we show the 3D objects.

Combining all these tasks, our minimum value product (MVP) would be a scene with 3D rendered planets and plants, where the player could add plants to the planet. Some

of our reach goals were to add a change from night to day, music, and potentially a watering mechanic for the plants.



(Figure 2: Initial sketch of the UI design and how the player would call each of the mechanics)

As we worked on the project, what we wanted from the scene and what we could do ended up changing. The final product includes majority of the MVP aspects, the changing planets, lighting and planting, however, we shifted from an animation of the plants growing to a changing of the planted model on each click of the plant. We also added some of our reach goals, such as revolving 'sun' or light source to simulate time passing, a texturing of the worlds and background, as well as asteroids to make the space feel more like space

---

2. Methodology

2.1 UI Design

When crafting the user interface initially tried to use Three.js three-rounded-box geometry to match the initial design. We wanted to use a tool that can function with clicks as well as rest ontop of the scene. The user would click on the planet that wanted to edit from the three buttons on the right and the flower that they wanted to place from the three buttons on the button left. On the fare bottom left, a larger version of the flower selected would display. The planet buttons would trigger a changePlanet function in Main.js that would delete the planet and render a new one. This then, deleted all the children of that planet. As we later decided to not have different flower to use at all times rather unique flowers for each planet, the interface on the left became obsolete.

The issue with the three-rounded-box was that we couldn't display images over the buttons. Instead, we rendered mini planets as buttons themselves to solve the problem. We did this still using the Box Geometry class like we did with the rounded button.

2.2 Tree Growth

To implement tree growth, we created three classes: leaf, branch and bush. Each tree growth method is initiated by the click handler function.

When a planet is clicked, a leaf and branch are initialized depending on which planet is selected. Rotation, position, and direction information is saved. Each component utilizes Threejs built in geometries. For example, Planet one's trees use cylinder

geometries for the base and icosahedron for the leaves.

When a tree is clicked, the animatedGrow function is called. Reduce the available complexity of the tree by one, and if it's non-zero, iterate over the number of segments in the clicked bush class. For each "parent segment", the number of children segments are calculated. New segments are generated based on the rotation, position and direction information stored in each segments parent segment. Each new segment is added to a THREE js group, and the group is rotated about y and z axises. The number of parent segments in the bush class is replaced by every new generated segment.

Per iteration of animatedGrow, leaf and branch dimensions are lowered.

2.3 Plant Placement

To physically place the plant on the planet, we needed to create a new mesh and move it to the position of the mouse click. This is relatively trivial since ThreeJS intersection objects give us the coordinate of intersection.

The more difficult part is orientating the plant mesh correctly so that it is facing "stem-up" from the planet. After looking through the ThreeJS rotation functions, we found a Quaternion method called "setFromUnitVectors" which sets the quaternion vector to the rotation required to move from vector V to vector U. By default, our plant meshes are placed with the stem facing in the global up direction (0, 1, 0). By setting V=(0, 1, 0) and U to the normal of the point-of-intersection on the planet, we can effectively rotate the mesh to the correct orientation.

2.4 Texture Mapping

We performed texture mapping on the individual planets and the background mesh that envelopes the world.

All textures were generated in Midjourney.

Applying textures to the planets was trivial. When the planet model was loaded in to a ThreeJS mesh object, we simply apply the texture to the mesh's material attribute.

To generate the "background" solar system, we first create a large sphere centered about the origin. The sphere is large enough so the camera is placed "within" the sphere and cannot escape it. The sphere is then textured with our solar system "nebula" texture to create a beautiful airy look.

2.5 Asteroid Generation

Every 5 seconds, a random spurt of asteroids will spawn in the world. The positions and spinning properties of these asteroids are randomized. The positions are set according to generating random points on a sphere, similar to how we simulated soft shadows on a point light.

Depending on their spawn position, the asteroids will then slowly move in a certain direction to make their way off-frame. The presence of scene fog makes it so the asteroids fade away rather than disappearing instantly once they clip behind the world sphere. After they clip behind the sphere, they are removed from the scene to prevent memory leaks.

2.6 Lighting

There are two lights in our game: an ambient light that provides background lighting and a directional light positioned above the planet that serves as the solar system's "sun".

The position of the directional light changes according to the update timestamp to simulate changing day/night cycles.

The material of the planets are set to a Phong material to emphasize the shading and specular reflections of the planet. The background sphere has a toon material as this material creates a visible shader line where the directional light points, further emphasizing the concept of "day" and "night" in our fictional solar system.

3. Results

We measured our results through rigorous visual testing of our features in a test-as-you-go approach. We also sometimes broke features down into smaller sub-features in order to test them.

When implementing flower placement for example, we first tested placement-on-click without looking at the mesh rotation at all. Once we confirmed that the mesh would be added in the correct location upon user click, only then did we start researching how to rotate the mesh correctly. (See figure 1 for results)

4. Discussion

Overall, we feel that our approach to this project was promising and that we parellelized our features efficiently. For example, when Darwin was working on the flower placement, he used the default flower model given in the sample code to test with. He didn't have to worry about Jack's flower generation process at all. Once both features were implemented, they were seamlessly combined in a single function call.

We all worked on our own branches and only pushed to main once we had individual features in a working state. There were inevitable merge conflicts here and there, but these were quickly resolved when we worked in person and each person could identity the necessary parts of their code that needed to be kept.

As we further progressed through the project, the design of our project started to drift away from the initial sketchups we created. Many aspects of the UI design in Figure. 2 where modified, and the resulting final product is a little bit different.

Instead of using the rounded-square-buttons for planet selection we opted to create a mini-version of the planet model as a "button" instead. We felt that this made the function of the switch-planet buttons clearer, and it allowed us to rotate the currently selected planet which we felt made the project more dynamic and fun.

Our initial idea of giving the user the ability to choose between different plant types also shifted. Since we had a bank of different planets to choose from, we felt it made sense to create different plant types based on the selected planet. For example, the cactus-like plant would only be able to grow on the desert-like planet. To accommodate for this change on the UI, we transformed the plant-switcher buttons on the UI to instead change the color of the sunlight shining down on the planet.

We encountered technical roadblocks and hurdles along the way, but we maneuvered

past them and kept the vision of our product flexible.

5. Conclusion

Overall, we feel that we effectively obtained our goal of creating a meditative zen space garden experience through the use of computer graphics techniques.

We showed our product to some friends outside the class and received positive feedback.

There are some quality-of-life issues that still persist. For example, clicking on plants that have reached their maximum generation potential will still generate a sound effect even though the mesh does not change. Sometimes, the random plant mesh generation can be a little glitchy and generate floating stems/leaves. We also detect mouse click with a pure onclick event listener. This means that if a user is scrolling on the planet to rotate it and releases the mouse over the planet mesh, a plant will be placed at the point where they let go of the mouse even though they probably didn't want to plant a plant there. Overall though, these don't detract too much from the experience.

There is a lot of room for expansion on this project. We could add different solar systems (different world sphere mesh textures), randomized "weather" events, and incorporate other peaceful aspects of nature besides plants (ponds, fish, bugs, critters, etc.).

Contributions

Samantha Trimboli worked on the UI design, 3D model collection, and changing planet mechanic. Raff Davila also worked on the UI design as well connecting the buttons to the major mechanic. Raff in addition, helped debug and texture the planets. Darwin Do work on the background textures/meshes, lighting cycles, asteroid generation, and flower placement. He also added music and composed interactive sound effects. Jack Li implemented the tree growth. We all worked together in order to combine our parts and discuss changes to the MVP/mechanics as we progressed throughout the project.

Works Cited

Planet and Asteroid 3D models
https://creazilla.com/nodes/5257-low-poly-planets-3d-model

Three.js three-rounded-button

https://www.npmjs.com/package/three-rounded-box

Sphere Point Picking

https://mathworld.wolfram.com/SpherePointPicking.html

Orthographic UI Example

https://codesandbox.io/embed/react-three-fiber-viewcube-py4db?codemirror=1