Giggling Gardens

A COS 426 Project by Sarah Bock, Eva Vesely, and Myra Norton

Link to the online demo: https://sarahmbock.github.io/cos426finalproject/

Link to the video demo: https://drive.google.com/file/d/1--aFBH2o4Lu5rlLAVNk25kY3Vb_ri-Xr/view?usp=sharing

SHORT ABSTRACT

Welcome to Giggling Gardens! Our game allows users to plant and water seeds that turn into a variety of different flowers. Some flowers are rarer than others and are therefore worth more points. The point of the game is to get as many points as possible before the timer runs out. There is also a small chance that a user might plant a “bad seed” that will terminate the game.

PREVIOUS WORK

Our project is a fun and effortless game for anyone who needs to take a break from their work to enjoy some virtual gardening. For such a simple idea, there are surprisingly few gardening games that result from a quick Google search. Of the few games that we found, none of them used 3D graphics but instead kept everything two-dimensional. The most impressive of the games that we found is called “Grow Your Garden,” which has a very cohesive design and a lot of smaller features that makes it feel complete. This game, however, is hard to understand and use if you didn’t first read the “How to Play” page. For this reason, we wanted to make the instructions for our game easy to access and the interface intuitive to use. Some games make the mistake of making things too complicated, and we wanted to make sure that our game would be as relaxing as possible. Another game, “Let Me Grow,” had a garden theme but didn’t let users plant their own garden. Unlike these two games, ours requires less brain power and encourages more mindfulness.

GOALS

We had a few goals for this project, and we fulfilled all of them! Our first goal was to make an aesthetically pleasing scene. We accomplished this by finding and incorporating a variety of different flower objects, creating a soil mesh, making our highlight mesh somewhat transparent so you can see the soil mesh, including moving clouds in the background, and adding a shelf to hold the seed bag and watering can. We also added some directional lighting and adjusted our other lighting sources quite a bit to get our objects and textures looking just right. Another goal was to make a user-friendly interface that is both functional and fun to use. We accomplished this by adding an instruction modal before the game starts, displaying the points and timer at the top of the screen, and adding a highlight mesh to show the user exactly where on the ground they are planting and watering. Additionally, when the user's mouse hovers over the watering can or the seed bag, the object appears to get larger and the cursor turns into a pointer, indicating that the user can click on it. On a larger scale, all of the members in our group wanted to find, read, and understand code from other projects and practice incorporating it into the infrastructure of our own game. We were able to do this by using various .GLB files from Sketchfab, learning how to make a grid from a YouTube video, asking ChatGPT to explain why some of our syntax wasn't working, using texture maps from 3dtextures.me, and borrowing a flower icon.

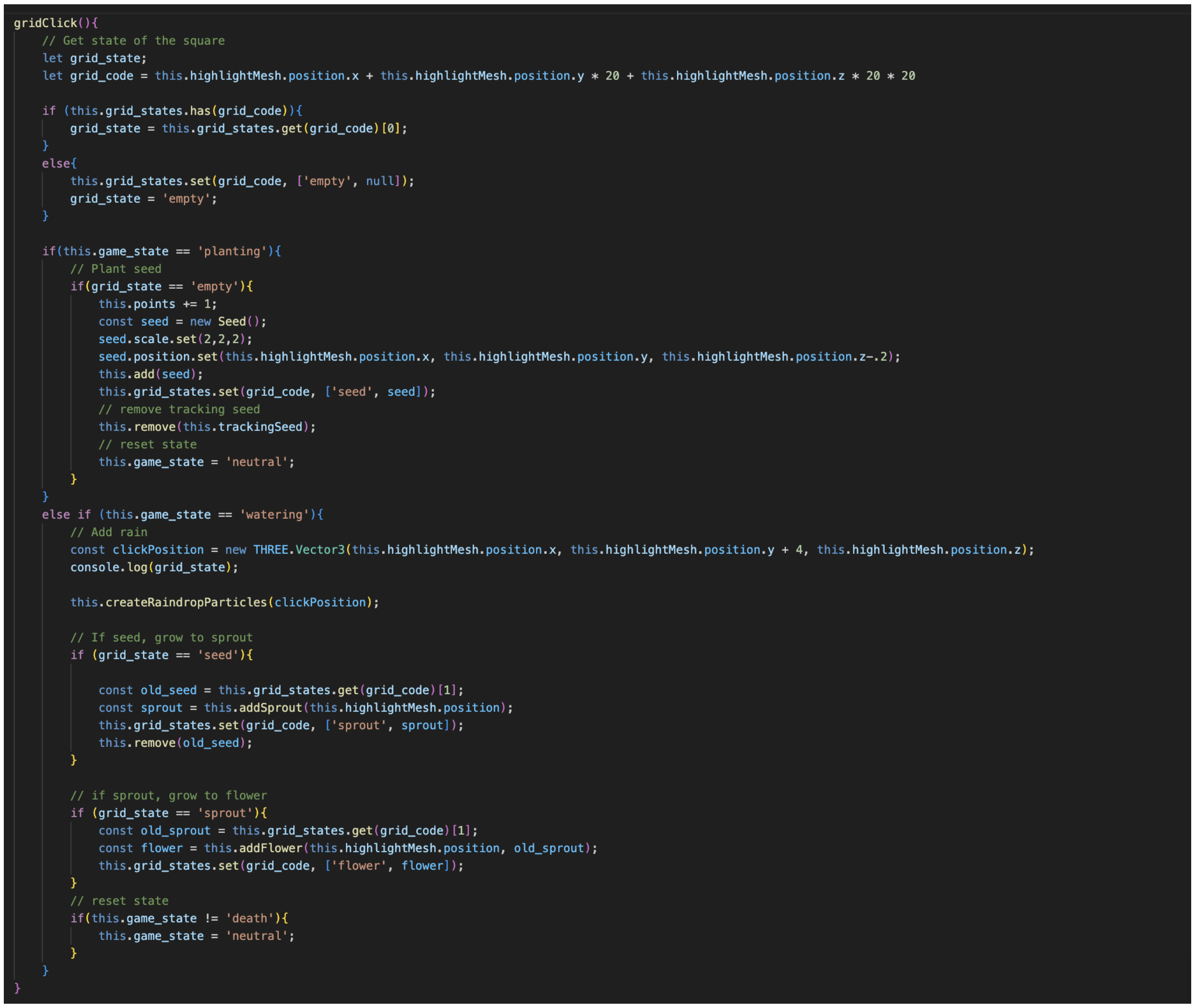
DESIGN DECISIONS

We decided to build our project off of the starter code provided by the course since it was concise and easy to understand. From the start, we wondered whether we should put the code from SeedScene.js into app.js. Combining the code would keep everything in one accessible place and minimize issues with importing different files. We decided to maintain separate files to keep our code modular. Keeping the files separate also gives us the ability to create more scenes if we continue to build out our game.

In general, we tried keeping as much logic about the objects in the scene, their appearance, and their movement within the SeedScene.js file. At the same time, the event handlers to catch interactions with elements on the screen such as mouse clicks and movements were registered in app.js. Because the interactions with the screen sometimes triggered visual changes to the objects, this created an ambiguity about where we should implement these features, and how to structure the relationship between these files in a principled way. We resolved this by creating a few methods within SeedScene.js that became attributes of the scene object which handled game behavior upon some user interaction. Within the event handlers in app.js, we simply call those functions to allow the meaningful logic to be implemented within SeedScene.

Another question that arose was that the expected behavior in response to some user events, like a click, sometimes depended on previous interactions. For example, if a user had clicked the watering can, the next grid click should trigger a different event than if they had just clicked the seed bag. We managed this by adding game state variables in SeedScene.js to keep track of the status of the game in between clicks. Additionally, for a given space in the grid, what should appear when a seed is planted or when the square is watered depends on what was already in the grid square. To keep track of this, we used a map. Based on the physical position of the square, we generated a unique code corresponding to the square to act as the key in the map. The value it pointed to was a list that stored what type of object was in the square and the

object itself. Using this map and the game state variables, we were able to accurately update the objects in the scene in response to user interaction.
This code below shows how we used the game and grid states to manage game flow.



```
gridClick(){
    // Get state of the square
    let grid_state;
    let grid_code = this.highlightMesh.position.x + this.highlightMesh.position.y * 20 + this.highlightMesh.position.z * 20 * 20

    if (this.grid_states.has(grid_code)){
        grid_state = this.grid_states.get(grid_code)[0];
    }
    else{
        this.grid_states.set(grid_code, ['empty', null]);
        grid_state = 'empty';
    }

    if(this.game_state == 'planting'){
        // Plant seed
        if(grid_state == 'empty'){
            this.points += 1;
            const seed = new Seed();
            seed.scale.set(2,2,2);
            seed.position.set(this.highlightMesh.position.x, this.highlightMesh.position.y, this.highlightMesh.position.z-.2);
            this.add(seed);
            this.grid_states.set(grid_code, ['seed', seed]);
            // remove tracking seed
            this.remove(this.trackingSeed);
            // reset state
            this.game_state = 'neutral';
        }
    }
    else if (this.game_state == 'watering'){
        // Add rain
        const clickPosition = new THREE.Vector3(this.highlightMesh.position.x, this.highlightMesh.position.y + 4, this.highlightMesh.position.z);
        console.log(grid_state);

        this.createRaindropParticles(clickPosition);

        // If seed, grow to sprout
        if (grid_state == 'seed'){

            const old_seed = this.grid_states.get(grid_code)[1];
            const sprout = this.addSprout(this.highlightMesh.position);
            this.grid_states.set(grid_code, ['sprout', sprout]);
            this.remove(old_seed);
        }

        // if sprout, grow to flower
        if (grid_state == 'sprout'){
            const old_sprout = this.grid_states.get(grid_code)[1];
            const flower = this.addFlower(this.highlightMesh.position, old_sprout);
            this.grid_states.set(grid_code, ['flower', flower]);
        }
        // reset state
        if(this.game_state != 'death'){
            this.game_state = 'neutral';
        }
    }
}
```

Finally, in App.js, we had to also add some state logic that wasn't based on specific interaction with the scene, but on whether the game should be running. We used a variable to keep track of whether the user had started the game or if they were still on the start or end menu to determine whether interactions with the game should be supported.

The most prominent object in our seed scene is the ground. The ground is composed of three different elements: the grass mesh, the soil mesh, and the highlight mesh. Both the grass mesh and the soil mesh are built using a plane geometry and a mesh standard material. The standard materials each take five different maps as input: base color, normal, height, roughness, and ambient occlusion. These maps were downloaded from 3dtextures.me to incorporate the grass and soil textures. The highlight mesh was also created using a plane geometry, but we only needed a basic material for it instead of a standard material. The purpose of the highlight mesh is to show the user which plot of land they are interacting with when planting or watering a seed. We chose to make the material of the highlight mesh somewhat transparent so that the user could see the ground through it. This made the highlight mesh act more like a “highlight” than just a solid plane that obscures the view of the soil. We rotated all of the planes so that they appeared to lay flat on the ground and then layered the meshes appropriately. More specifically, we put the grass mesh below the other two meshes since it is the largest (50, 50) and should be in the background of the scene. The soil mesh was laid on top of the grass mesh and covers a smaller area (20, 20). Lastly, we put the highlight mesh on top of the soil mesh. This mesh had to be on top otherwise it would have been obscured and would not have provided its intended function. The highlight mesh is the smallest mesh (1, 1) and is initially set to be in the middle of the screen. However, and event handler in app.js tracks mouse movements and updates the position of the highlight mesh.

A user will find that they can change the camera position to look around the scene. We decided to keep this feature so that they can fully experience the three-dimensional space of their garden! However, we decided to limit the extent to which they could look around so that they couldn’t go under the plane mesh or spin all the way around horizontally. We did this by bounding our orbit controls.

HURDLES ENCOUNTERED

One of the biggest hurdles we encountered was debugging the weird discoloration we encountered when loading our ground texture and various objects. This issue became apparent as soon as Myra started loading in the different maps required to create our ground texture. All of the images we tried appeared way brighter than they looked online. Then, we noticed that a lot of our objects were rendered a lot darker or more saturated than they appeared on Sketchfab. We were pretty baffled by this for a while. Eventually, we loaded a texture into the source code for one of the YouTube tutorials that we were following and saw that it looked normal. We decided to copy over the exact lighting parameters from the source code and viola! Our ground texture looked perfect. From there, we started playing around with our lighting some more to get a scene we were happy with. We realized that our main issue was that our hemisphere light was way too intense. We also switched our spotlight with a directional light because it seemed to have more of an effect on the scene. These changes fixed the appearance of our ground texture and the over-saturation of some of our .glb objects, but we still faced the issue of some objects appearing too dark. After searching online, we realized that the issue could be the metalness property of the object mesh, which needed to be set to 0. This fixed the issue for one of our flowers but not the other objects. After inspecting the mesh properties of the other objects and trying to fiddle with the parameters, we eventually decided to just switch the wonky objects out.

One of the most time consuming parts of this project was implementing the soil mesh. Although the code for this was relatively simple, there was a lot to learn and debug. Our first implementation was a plane that took an image as a standard material, which obviously didn't look great since it was flat. We wanted to add some bumpiness to the plane, so Myra generated her own displacement maps, but those didn't look great either since they didn't align with the image we were mapping onto the plane. Finally, we found textures online that came with all of the necessary maps. While this implementation had the texture we were looking for, the color was heavily saturated. We spent a long time trying to debug this until we realized that it was the lighting issue described above.

GROUP DYNAMIC

In making our game, we all gained valuable insight into how we work in group projects. We were all relieved to find that we worked incredibly well together! In the beginning, we were able to meet often to discuss our initial ideas for the game and the overarching structure of our code. Once we had established our vision for the game, we sat together and used the liveshare feature on VSCode to piece together the foundation of our program. Once we had a codebase that we had all contributed to and understood, each team member was able to independently implement a reasonable number of features that suited their skillset. We had intermediate check-ins where we could explain how our code changed certain functions within the game and how that would impact the features that other members were implementing. Each team member has written a paragraph outlining the contributions they made to the project, which are included below:

Myra - At the beginning of this project, I setup a basic grid helper and highlight mesh in SeedScene.js that interacted with an event handler in app.js. I provided the basic skeleton for the ground and the initial event handler that puts a flower on the grid square when clicked, which Sarah later expanded. My next big task was implementing the soil mesh, which is explained in greater detail in the section outlining the hurdles encountered. Once I had a handle on implementing textures, I added a grass to the background of our scene. To further beautify our scene background, I added some cloud objects that move with the wind. The cloud that is positioned further from the camera travels across the sky at a slower speed than the nearby cloud to more accurately simulate their difference in depth. I added the instructions and game description to the modal that appears before the game starts and worked on the refresh button that Eva debugged and included in the modal that appears when the game terminates. Next, I made interactions with the seed bag and watering can more intuitive by enlarging the objects when you hover over them and changing the cursor to a pointer. After doing this, I actually changed the seed bag to one that matched the aesthetic of the watering can and added a flower that also better matched the aesthetic to make for a more cohesive scene. Lastly, I changed the title icon. After all, the devil is in the details!

Eva - I started the group off by figuring out how to load an additional object into the scene. I created a folder for the watering can by copying the folder for the land object and changing it to implement the watering can class. This was when I realized that the objects needed to be .glbs rather than.gltfs, and used a website to combine the watering can .gltf with its .bin file. I also helped Sarah set up the game states and grid data frame to hold each grid square state. When working on my own, I cleaned up the group's code, solidified the probabilities/ points associated with each flower, formatted the start/ game over menu, added a bad venus fly trap flower which would immediately end the game, and made the sprouts spin when they first grew. I also added game phase logic in app.js so that users couldn't interact with the game when the start menu was present. Finally, I helped Myra figure out the lighting/discoloration in the scene and replaced the seed object to fix some of the discoloration.

Sarah - I worked a lot on implementing the basic functionality of the game and designing the code structure to handle the game workflow. I determined how to divide the work between App and SeedScene so that event handlers in App.js would trigger the desired impact in the scene. This entailed adding the watering can and seed bag objects to the scene (though they were eventually replaced with nicer-looking ones), and defining game states such that a user could interact with the objects to plant and grow seeds within each grid square. Once the fundamental logic was implemented, I worked on making some improvements to the display to make it more intuitive for users. This included making the watering can and seeds track the mouse around the grid while watering and planting respectively, and adding a rain animation after a plant was watered.

NEXT STEPS

We could work on fine-tuning this assignment for a long time! Right now, there is grass outside the fences that is implemented as a plane mesh with a mesh standard material. It could be cool to build up the grass so that it's more realistic and sways in the wind. Additionally, the animation for watering the plant could be cleaned up a bit so that the watering can rotates and the raindrops fall more naturally. One of the more involved next steps could be figuring out how to dig a hole. It would be relatively easy to import a shovel, and we could create a hole by adding an object that looks like a hole or by removing a piece of the mesh.

Some bigger next steps for the game aspect of our project (as opposed to the graphics) would be to add the ability to kill a plant. Perhaps if each seed requires a different amount of water, you could kill it by not watering it fast enough or watering it too much. If we were to implement this, we would need to keep track of how much water the plant has received, which wouldn't be too different from how we keep track of the state of a grid square or the state of the game. Another way to further gamify our project is to motivate the user to unlock different types of seeds. This makes the garden more customizable instead of relying on chance to determine the type of flower produced. Implementing this feature would give the user more control over the game and therefore make it more enjoyable since it is frustrating that you can die at any point with a bad seed. Lastly, it would be fun if the user could be rewarded with other garden scenes once they unlock all the seed types possible for that garden scene.

CITATIONS

Previous work:
https://jrob774.itch.io/grow-your-guarden
https://www.groplay.com/apps/grow-garden/
https://www.abcya.com/games/let_me_grow

Website with .glb and .gltf files:
https://sketchfab.com/search?features=downloadable&q=watering+can&type=models

Website for combining .gltf and .bin files:
https://glb-packer.glitch.me/

Website with texture maps:
https://3dtextures.me/