3D Snake: Writeup

By Rachel Inman and Archana Sharma
COS 426 Final Project
May 12, 2020

Abstract

For our final project, we decided to implement a 3D version of the classic Snake game in Three.js. In this version, you are a little red spherical snake in someone's front yard. Eat the mice to get bigger, but don't hit the fence! The game is played in first person for more ~intrigue~ and an extra challenge as well.

Introduction

Goal

We tried to make a unique spin on the famous Snake game. Historically, Snake is a 2D and minimalistic game played in third-person where the snake runs around on a flattened plane. Wouldn't it be fun if Snake was instead in the 3D world? Our original conception was to make a colorful cube that the snake would fly around in, eating items in the 3D space. However, this implementation did not work well (see **Introduction.***Approach* for more information). Instead, we decided to make a 2.5D version, where the world was 3D and the game was still played in first-person, but the snake still navigated a plane, not a cube. Thanks to COS 426 user feedback for the 2.5D idea!

One primary goal for our implementation was ease of playability. Making a first-person version of Snake is a significant departure from how the game is normally played, and so we wanted to make sure that the first-person controls were intuitive, easy enough to use, and allowing for full range of gameplay.

Another goal was visual appeal. We were trying to make a game, so we wanted the experience to be appealing. Even if the ultimate implementation was minimalistic, we wanted it to be sleekly and attractively minimalistic, not barren.

Another goal was setting our version of Snake apart. We wanted to create something unique. Although the idea of 2+D, first-person Snake was already unique, we wanted to make the game distinctively ours in some way.

Previous Work

People have already made dozens of Snake games before, so the fundamental idea we wanted to build on was not unique at all. This was a good thing in many ways, as it gave us material to consider. The generic Snake game gave us the ideas of ease of use, items to eat, collisions with self, and not going beyond the boundaries of the plane.

Another piece of helpful user feedback was directing us to slither.io, a website that features a unique take on Snake. Slither.io is particularly unique because it is multiplayer and

you die from colliding with other snakes, not yourself. But slither.io also has controls where the snake moves fluidly and is directed by the mouse, not arrow keys. This was food for thought for us when we were trying to implement the 3D version of our project and when we were thinking about how we wanted to implement first-person controls in the 2D version.

Approach

Prior to the version that we submitted, we tried making a 3D version. This consisted of a giant cube for which each of the 6 walls were colored differently. Then, from the perspective of the snake, you could use the mouse to fly around fluidly in the 3D space, similar to the motion of slither.io. We tried this using the FirstPersonControls and FlyControls in the Examples section of the Three.js docs. However, it ended up being very difficult to navigate this space, as well as incredibly dizzying. After several failed attempts to make this most-3D version work, we decided to move onto something that would have a better user experience and be less difficult for us to implement.

Our next approach, and the one that we ultimately stuck with, was making a 2.5D version with a first-person point of view. We thought that this would be better for two main reasons. For one, this approach would be much easier for us to implement. Faced with time constraints and relative newness to Three.js, we decided that making a more usable, polished 2.5D version of Snake was the better choice for this project than making a barely-functional 3D version in the same span of time.

The other reason we decided to go 2.5D is because this experience was far better for the user. Even when the movement was okay, it turned out that flying through space in all directions is a dizzying experience (evidence: Rachel legitimately felt sick for 2 hours after trying to debug the 3D version at one point). We realized that Fly Controls is best for when a user is flying *through* space in one principal direction, like first-person flying games generally are. This was different from what we were trying to use it for: going in all directions at a fairly rapid speed.

The other main choice was making the game first-person. It was entirely possible to make a 2.5D game with a third-person perspective, and this is what we did at first. This was fun and in many ways like the classic Snake game. But that was also a problem because we wanted something of our own, and some of the fun of the original 3D idea was from the first-person perspective. So we decided to make a 2.5D, first-person game.

Methodology

The Plane

We wanted to make a plane for the snake to move around on to keep it relatively consistent with the original Snake game. In the original 3D version we were planning, there would be no plane. It turns out that this plane is very grounding and clearly demarcates the boundaries of the snake's world.

We implemented this with a simple Three.js PlaneGeometry structure, 100x100. In order to make it like a "floor" we rotated it by (- Math.PI / 2) so that it lay flat on the y-axis. We gave it a Phong material so that it could reflect light. We also gave it a texture so that it could look like grass, in alignment with our ultimate decision to decorate the game like a front yard. We created

this texture with the help of the following texture maker: https://mebiusbox.github.io/contents/EffectTextureMaker/.

We could have implemented an infinite plane, but we did not. If we had, this would have taken away from the challenge of the snake colliding with itself and would also make decorating more difficult.

Lighting

For lighting, we kept it simple and followed the structure given by Reilly Bova's skeleton project in the BasicLights class (https://github.com/ReillyBova/three-seed). We added a Three.js AmbientLight to softly light the entire scene, and we also used a Three.js DirectionalLight to imitate a sun. This approach lit the game up nicely, which is what we wanted.

The Snake

For obvious reasons, implementing a working snake was important for our Snake game. We made our snake in *Snake.js.*

Snake object: Stored references to the scene's camera *camera*, the absolute direction the snake was moving in *direction*, and an array of the snake's segments *segmentList*. The constructor initialized the snake and had a *document.eventListener* to listen for "keydown" events.

addSegment(): When called, it would generate a new sphere (the 3D shape representing a segment of the snake's body) with SphereGeometry and MeshPhongMaterial. If *segmentList* was empty, the "head" sphere's position was initialized at (0, 1, 0) in the center of the plane. Otherwise, the segment was appended to the existing snake by adding the sphere at the location of the last ("tail") segment stored in *segmentList.*

onDocumentKeyDown(): To set the snake's direction, this function took in a keyboard event as a parameter. If the keypress wasn't from an arrow key, it was ignored. Otherwise, if the snake wasn't moving (aka *direction = 0)*, *direction* would be set to the direction of the arrow key. Otherwise, if the snake was moving, the snake would switch directions in a way relative to its current position. So, for example, if the snake was moving absolute left and the right arrow key was pressed, the snake's direction would then switch to absolute upwards. This was to make a realistic first-person navigation. This code was loosely inspired by

moveSnake(): The function responsible for actually moving the snake. It iterated through each segment in *segmentList.* If the segment was the head, the head would move in the direction specified by *direction* by a distance of *segmentDist* (arbitrarily defined as 1.5). If the segment wasn't the head, then its position shifted to the space of the segment in front of it. This gave a convincing illusion of movement.

**When implementing *onDocumentKeyDown()* and *moveSnake(),* the code was loosely adapted from / inspired by some parts of this Three.js example: https://threejs.org/examples/misc_controls_pointerlock.html. Specifically, the keypress eventCode switch statements, picking up on *"keydown"* with an *eventListener,* and the overall structure of the approach were particularly helpful.

moveLeft(), moveRight(), moveUp(), moveDown(): Helper functions to streamline *moveSnake().* These updated *direction* to reflect the new direction of movement, moved the snake head in the appropriate direction by *segmentDist,* and adjusted the camera to move with the snake's head. They also pointed the camera in the right direction. To do this, the camera would look at the spot on the edge of the plane it was moving towards.

Adjusting the snake's direction did not actually move the snake. Instead, *moveSnake* was called repeatedly by a function in *SeedScene.js.* The snake was added in *SeedScene.* After this, the function *window.setInterval* would call *moveSnake* repeatedly every 100 milliseconds. You could adjust the time for a faster or slower snake.

We picked this implementation because it was straightforward and decently modular. The one awkward thing was that in order to get the scene camera, we passed *camera* as an argument into *SeedScene.js* from *app.js,* and then we passed *camera* from *SeedScene.js* to *Snake.js.* This was definitely a little ugly and made Rachel long for React props and top-down flow, but it got the job done effectively.

The Mice

The mouse object appears in a randomly selected location at the start of the game. When a collision with the snake is detected, the mouse disappears and reappears in a new random location on the plane. This continues until the game ends. At each collision, the stored score increments.

The Models

In order to make our code look pretty, we populated it with some models. We decided to make a scene of a house's front yard, because that seemed pretty. We got a house model, some fence models to border the yard, a pool, and some trees. This made the scene much less bland than its original implementation while also letting us focus on making the gameplay.

Credit for the models is as follows:
* [Farmhouse](https://poly.google.com/view/bHyQe5jzdiQ)
* [Fence](https://poly.google.com/view/8r5ZAEhrppD)
* [Mouse](https://poly.google.com/view/6DOjEGKd8nx)
* [Pool](https://poly.google.com/view/bHyQe5jzdiQ)
* [Tree](https://poly.google.com/view/68OOL4zL6Co)

Collisions

At each movement of the snake, we check the current location of the snake head for any collisions with the fence boundaries or the mouse. If a collision with the mouse is detected, the mouse disappears and reappears somewhere else on the plane, and the snake is lengthened by one segment. If a collision with the fence occurs, the game ends: the snake stops moving and a score is displayed.

What Didn't We Implement?

There were several things we would have liked to implement but did not. One of these is moving obstacles or making the models in the scene themselves obstacles. This would have been a cool feature and provided an additional challenge, but it was also challenging for us to implement and we were running out of time.

Another thing we might have implemented is smoother first-person controls. Personally, the "clunky" first-person controls with their harsh 90-degree camera rotations were actually somewhat enjoyable for us - they added some degree of challenge and kept the controls simple and straightforward. Nevertheless, if we wanted to experiment, in the future we might have tried making more fluid controls with mouse control or something similar. We did not try this feature out because of lack of time as well.

Finally, a simple user screen would have been good. This would have allowed us to see the user's score, as well as to hit "game over," "play again," and the like. We did not implement this because of lack of time.

Results

Success Parameters

We measured success by the three things we were aiming for: playability, visual appeal, and uniqueness. The game needed to be playable and have basic functionality; it needed to be attractive; and it needed to add something new to the classic Snake concept.

Discussion

Is the Approach Promising?

On the whole, we think our approach is fairly promising. In particular, we liked the idea of a first-person game of Snake. We're not sure if this has been done before by anybody else, but it's certainly a fun way to jump into the game, and it adds an extra level of complexity to tail collisions. In terms of our implementation, our approach was generally modular, clean, and straightforward. Although certain aspects could be improved upon (for example, passing down the camera to *Snake* was a bit of a faux pas), our code worked and is decently legible - that was most important to us.

Our fully 3D idea is also promising, but much more difficult to implement, as we learned. Any good 3D Snake game would have to have excellent user controls and design to function.

Conclusion

How Effective Was It?

On the whole, we think that the results were fairly decent. We got a late start to the project after trying unsuccessfully to make a fully 3D version, so getting anywhere was good on the playability front. We managed to implement first-person controls with relativized direction input from the arrow keys; the snake moved and could grow; it moved on its own; … So many of the basic and intuitive playability aspects of Snake were there. And no dizziness from 3D!

In terms of visual appeal, we think that adding models, textures, and lights really helped make the game look decent. In our MVP, the snake was just a sphere moving on a plane. This was fair enough but excessively minimalistic. Adding mice to eat, trees to drive around, and a little scene helped make the game feel like home.

We think that our idea for the Snake game was relatively unique. Although it was a step down from our original idea in that it was not fully 3D, it still had 3D elements. Most notably, it was in third person. Its visual effects for eating mice and being in a backyard also added a personal touch.

Next Steps / Issues to Revisit

Most of all, our game could benefit from more time and energy spent on it. If we were to spend more time on it, we would focus our attention on smoothing out first-person controls and working on intriguing collisions. We would make sure there is no lag in the first-person controls and make sure that the abrupt rotation of the camera with arrow keys is the best approach (as opposed to fluid motion of the camera with the mouse).

We would also work on intriguing collisions. We were thinking of making spontaneous collision objects that are randomly generated - perhaps they could be models like a fox or another natural enemy to snakes. We could then also make the snake die ending the game upon collision with these predators or other obstacles. If the other objects were lethal, then we would need to make sure that the food-mice were not generated inside those meshes.

What Did We Learn?

We learned that making fully 3D versions of games is actually very difficult and the controls can be mind-bendingly dizzy. We learned that Three.js can be unintuitive and small things can take a while to figure out. We learned that making games is fun! This has ignited our desire to improve upon this game and continue improving on our implementation.

Contributions

Rachel Inman: MVP (i.e., snake ball, plane, lighting), *Snake.js* implementation, environmental/decorative models, first-person control of snake

Archana Sharma: Mice/point generation, collision detection (between snake and mouse, snake and fence), updating score, displaying score/ending game

Works Cited

(see above for where and how these were used)

https://threejs.org/examples/misc_controls_pointerlock.html
https://mebiusbox.github.io/contents/EffectTextureMaker/
https://github.com/ReillyBova/three-seed
https://poly.google.com/view/bHyQe5jzdiQ
https://poly.google.com/view/8r5ZAEhrppD
https://poly.google.com/view/6DOjEGKd8nx
https://poly.google.com/view/bHyQe5jzdiQ
https://poly.google.com/view/68OOL4zL6Co
https://poly.google.com/view/6DOjEGKd8nx