COS 426 Final Project: Princeton Jenga

Tiffany Deane, Caroline Hana, Joseph Lou

Abstract

We aimed to implement the classic game Jenga, with a slight Princeton twist to show our irritation with the construction that has been going on for years. We therefore used construction-themed elements in our game, such as a background image of Yeh College still under construction, and Jenga blocks as steel beams. We implemented some additional features, such as adding sounds, allowing the user to rotate the view of the tower, a top view, an on-screen control panel, textures (as mentioned), and the simulated dynamics of gravity. Unfortunately, we were unable to get the collision detection working properly, which means our game is currently unfinished.

Introduction

Goal

We aimed to implement the classic game Jenga, with a slight Princeton twist to show our irritation with the construction that has been going on for years. We therefore used construction-themed elements in our game, such as a background image of Yeh College still under construction, and Jenga blocks as steel beams.

Previous Work

We found an existing version of 3D Jenga built with THREE.js and a physics library Physi.js: http://chandlerprall.github.io/Physijs/examples/jenga.html. However, there were missing features with this version that we aimed to add, such as the ability to move the camera around, a scoring system, game over detection, background music, and more.

Approach

We used the provided starter code and attempted to combine it with the example base code we found. In theory, it was simply a case of integrating the physics library, then adding all our additional features on top.

Methodology

The features that we wanted to implement beyond the existing code were:

- Rotating the table and tower to view from different angles
- Top view

- On-screen control panel with current score and player's turn
- Background music and sounds of blocks hitting each other
- Textures for background, table, and blocks

To rotate the view, we were able to use the starter code, which provided a THREE.OrbitControls example for how to allow the user to use their mouse to change the position of the camera. We disabled vertical motion while keeping horizontal motion, so the user could view all around the tower from a fixed vertical angle.

For the top view, we added another THREE.Scene that inherited all the children of the main scene so that all elements would render for both of them. We fixed the camera position for the top view so the user couldn't control it, and used a different background to differentiate it. The top view sat at the top right of the screen.

The on-screen control panel displayed some controls for the user, such as using the spacebar to pause the game, "m" to mute, and "r" to restart. It also included a scoreboard that kept track of how many blocks the player had removed.

We found some background music and sounds, and triggered them upon certain events, such as starting the game, muting or unmuting, and placing blocks.

Drawing from the base code, we used a texture for each THREE.Mesh that represented a block. We also added appropriate backgrounds to the THREE.Scenes that were in use.

Although we wished to use the example base code, we were unable to make the Physi.js library work. (It seems that the code hasn't been maintained in years, and some other people have experienced similar issues.) Instead, we decided at the last minute to implement gravity and collision detection ourselves, but we were also unable to eventually get that fully working. While the blocks are pulled down due to gravity, they don't collide with each other, and therefore we don't form a full block tower.

The approach we took seemed promising, but it eventually did not work out. We definitely tried to compromise with our situation by switching to doing the collision detection ourselves, but we were unable to finish it with the time we gave ourselves. Perhaps we shouldn't have tried so hard to integrate the library with our code, which used up a lot of the time of our project.

Although the game didn't work at the end, it was still a fun experience, and we all learned a lot. It was some of our first times building a game, especially one that runs entirely in a browser, and working with all of the components was very interesting.

Conclusion

For next steps, it would obviously be great to get the game working. This would involve fixing the Block.js file's handleCollision() method, which checks for and handles the collisions between two Block objects. We also have the applyGravity() method, which works, but seems a little bit buggy.

We also had a lot of stretch goals that we wanted to implement but didn't have time for due to the collisions issue. For example, our game is currently single player, as we had not yet gotten around to adding turns. The way users interact with the tower is also not completely fleshed out, and the animations are not fully working either in terms of putting a block back on the top of the tower. These are all things that we would have liked to improve with more time.

Contributions

- **Tiffany:**
    - Implementing block geometry with THREE.js
    - Tower creation with THREE.js
    - Block textures
- **Caroline:**
    - Start screen page
    - Block sounds
    - Screen control panel → restart game button, number of blocks removed
    - Arrow key selection and removal of blocks
- **Joseph:**
    - Main game view controls: only allow left and right
    - Top view
    - Background textures
    - Page transitions / keypress event handler
    - Playing / pausing audios upon certain events, plus muting

Works Cited

1. THREE.js documentation: https://threejs.org/docs/index.html
2. Final project starter seed code (provided): https://github.com/ReillyBova/three-seed
3. Physi.js library: https://github.com/chandlerprall/Physijs
4. Example Jenga game: https://github.com/chandlerprall/Physijs/blob/master/examples/jenga.html