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

At the ideation phase of our game we knew we wanted to create a 3D version of a pre-existing game we all enjoyed. We first considered games on GamePigeon or iPhone messaging games, but realized a lot of them have been done previously. Inspired by the construction all around campus today we instead decided to implement a 3D online version of the board game Jenga. The classic game Jenga, with a slight Princeton twist to show our irritation with the construction that has been going on for years. We thought this game would be an interesting challenge since it is not originally a virtual game and incorporates a lot of real world physics. Additionally, the game Jenga is a simple game of stacked wooden blocks, and so we had a lot of creative liberty to build upon it for additional features and our own storyline. Specifically our storyline being around Princeton themes and the current state of campus, we thought would appeal to Princeton students as it reflects general campus opinions.

As our basis to the project we found an existing version of 3D Jenga that was built using THREE.js and a physics library Physi.js: http://chandlerprall.github.io/Physijs/examples/jenga.html. This github essentially is meant to demonstrate a physics library called Physi.js created by Chandler Prall through game examples that demonstrate gravity and collision, one of them being Jenga. The Jenga game created presents a minimalist working version of the game, where there are 48 blocks stacked 3 to a row on a table similar to the real life game. To play, users are able to click and drag on any block with their mouse to remove it from the stack. The game replicates realistic gravity and collision conditions where if a block is removed poorly or if one is removed that disrupts the structural integrity of the stack, the rest of the stack ends up falling as well. There are, however, a few drawbacks to the game. Firstly, the mouse control of clicking and dragging the blocks makes it very difficult to remove blocks carefully, as a result the stack falls during the first round of the game. Secondly, even if a block is successfully removed, there is no mouse or keyboard control to move the block up on the z axis and place it at the stop of the stack to end the turn. The lack of these two features make the game quite unrealistic and far from the rules of the real game. Apart from this, there were missing features with this version that we believe would make the game more enjoyable and aimed to add as well, such as the ability to move the camera around, a scoring system, game over detection, background music, and more.

We used the provided starter code provided to us by the COS426 course staff as well as the base code Jenga game found on GitHub in attempts to combine them and build upon it using Three.js. We used JavaScript event handlers for the user controls and referred to the code from Assignment 5 to implement textures and simple simulated dynamics. In theory, it was simply a

case of integrating the physics library, then adding all our additional features on top. Our MVP goals were to have the option for single or multi-player games and to have users be able to select and remove blocks from the stack as place them on top without letting the tower fall, otherwise the game would end. Additionally, we had goals to set up a score system, textures and sound effects to make it more appealing.

Methodology

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

- Better gravity and block simulation
- 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
- Collision Handling for when the tower fell or blocks hit the table

Multiple Views:

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.

User Controls:

Initially for user controls, we aimed to be able to select and drag blocks from the Jenga tower using the mouse and be able to move them up as well in the z direction to place them at the top of the tower. Similar to the Jenga base code, the usability of mouse clicks for the game was a poor system and not ideal for game play. Instead, we decided to have users play using the arrow keys. In this way, users could use the left and right arrow keys to highlight a block on the tower and press enter to select it. By selecting the block it removes it from its original position and places it at the top in the right orientation. This selection and removal process was done with event handlers and placed at the top appropriately with the correct coordinate calculations that account for the number of rows and blocks in each row. Additionally, at initialization of the block tower, the individual blocks were pushed onto an array and so each block has an associated array index, helpful for block selection as well. This process of block selection and removal due the use of an array data structure which made it difficult to keep track of block indices when they are rearranged and placed on the top of the the tower during game play. This causes issues when removing blocks since the wrong block would be removed or several blocks would be removed at one from the tower. This could probably be resolved by choosing a different data structure such as a map or key-value pair system, where each block has a unique key for identification and the coordinates of its position on the stack as the value, but this could not be implemented in time.

Additionally for user control an on-screen control panel was created to display some game controls for the user. Firstly, to use the spacebar to switch screens from the start menu to the game screen. Other controls were written at the top left of the game screen, stating controls

such as using the spacebar to pause the game, “m” to mute, and “r” to restart. This section also included a scoreboard that kept track of how many blocks the player had removed as they pressed enter reach time.

Sounds:

For the sounds in the game, we found some background music that played during the game screen and also switched music when in the paused screen, Additionally a sound effect was added upon triggering certain events such as placing blocks, so that at the press of the ‘Enter’ key you would hear a metal sound to replicate the noise of two steel beams hitting one another. Further a feature to mute and unmute the game’s music and sound effects was added as well.

Textures:

Drawing from the base code, we used a texture for each THREE.Mesh block, specifically grabbing a steel texture from online and applying to each block with MeshLambertMaterial(). As a result, all the blocks had a reflective look of metal to give it the look of steel beams, in theme with the construction game. We also added appropriate backgrounds to the THREE.Scenes that were in use, such as an image of construction happening for the new residential colleges and the start game scene..

Simulated Dynamics(Gravity) and Collision Handling:

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 based on the Assignment 5 code. Within Assignment 5 we implement gravity particle by particle and so instead we assumed each block was a particle in our case. Firstly this meant making each block an object, to establish these physical properties on each one of them. For applying gravity forces we added vectors of acceleration and damped velocity to the position y vectors. Furthermore, to handle collision between blocks and the table, the block object has a handleCollisons() method where a bounding block is established for each block on the x, y, z planes. As we iterate through the block objects in the main game file we check for collisions especially in the placing of the blocks. In such a case, the y positioning of the two blocks considered must not interest and should have a buffer between them.

While the blocks are pulled down due to gravity as we see them fall after removal, they don’t collide with each other, and therefore we don’t form a full block tower. Additionally, the blocks have issues with collision with the table causing it to fall through the table as well. 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.

Results

We were able to produce a slightly functioning game with most of the features we had aimed to implement, but unsuccessfully implementing collision handling tarnishes most of the game play.

Our metrics for our game were mainly qualitative in the form of user experience, how playable it is and the growth from the original base code. One of our main sources of debugging

had been from handling the gravity of each block, block selection and collision/ bounding box of each block. Due to these bugs having significant impacts on our game play it does indicate poor results. However, several other minor aspects of the game work successfully being the texturing, user control and block selection, sounds, perspective viewing, and even the start, pause and game over scenes. These features being included and implemented curate the user game experience and make it feel like a professional playable game. Even the blocks being able to fall with respect to gravity is a massive improvement from the original version of the game and more realistic. Despite not having perfect game play, we think our game exceeds the base code and has made it a more user friendly, interactive and interesting game for players.

Discussion & 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 collision 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. Essentially we would need to revisit our block mapping data structure and methodologies for implementing gravity and collisions on blocks, possibly incorporating friction and bounding box restrictions. It may have been an over reach to try to implement a game that simulates real life physics and Newtonian laws.

We think either way that this game developing experience has been very useful for learning how to intertwine many of the features we learned this semester. Initially, it would have been better to research the libraries planning to be used beforehand to understand how it can be integrated into our coding environment, or to check to see if other developers have issues with the library (which we noticed in the filed github issues) Despite this, this has been a good experience to learn how to problem solve and develop quick solutions based from previous assignment code before as we encounter new bugs in our code. 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.

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