Vincent Etherton and Aidan Ward
COS 426
December 13, 2024

COS 426 Final Project Writeup – Dynamic Pinball

Abstract:

Dynamic Pinball is a game that leverages the Three.js library and Cannon.js physics engine to create a dynamic, infinitely generated pinball experience for the user. The aim of the game is to control the pinball paddles to propel the ball forward into procedurally generated piles of cubes. The run ends when the ball — typical to conventional pinball — passes through the gap in the paddles. Dynamic Pinball makes use of various TypeScript classes to accomplish this game architecture that provides endless demolition fun.

Introduction:

Infinite runner games are very popular time-passing games providing an endless environment usually involving avoiding objects while collecting points. Subway Surfer and Temple Run are a few modern examples of these infinite runner games. Dynamic Pinball is a TypeScript-based application built using Three.js and physics rules from Cannon.js where the player controls two pinball paddles to continually hit a pinball up an upward slanted table. Rather than a typical pinball table which might have ramps, bumpers, or sensors to determine when the player scores points, Dynamic Pinball generates randomly noisy piles of cubes as the table surface 'slides' towards the player's view. For each cube that is displaced by the player's actions, the player should be awarded a point. Our goal was to create a fun, interactive, and hectic game of destruction and displacement while harnessing the nostalgia of playing on a classic pinball table.

We drew from one source when considering how to enhance our original idea to pursue a pinball-based game. Flying Terrain Generator, built by Pierce Maloney and Mitchell Cooper from the 2022 Art Gallery, provided the idea of an infinitely generated game environment replicating the common infinite runner format. Their approach to infinitely moving terrain with chunks influenced a similar chunk structure in which we generate random piles and their use of noise to generate mountain peaks inspired making randomly noisy piles. We wanted a greater sense of interactivity with the environment, so we chose to leverage a physics engine so that we could physically control the pinball with paddle hits and collide with objects generated on the table. With this new physics engine, we could no longer implement the chunks as a ring structure as this past work did and instead made it a sliding generator, which will be discussed later.

Our approach consisted of creating a table surface and bumpers with physical properties governed by Cannon.js and its ability to create a physical world. The piles are generated chunkwise, and move towards the user's point of view to set up pinball collisions with the piles of cubes. Creating many classes representing the different objects in our scene allows for the instantiation of multiple pile objects and cube objects all for the user to interact with.

Methodology:

User Controls:

The user controls the two paddles at the bottom of the screen with the Q and P keyboard buttons to control the left and right paddle respectively. We utilize event handlers to accomplish

this. While each key is pressed, a Cannon.js impulse force is applied to the back of the respective paddle body at its end to simulate a quick hit of the paddle. This on its own would not be enough to create the expected swinging motion of a pinball paddle, however, as it would simply keep moving the paddle in the direction in which the force was applied relative to the paddle. To resolve this, each paddle is constrained by a hinge to an immovable physics body to create the rotating effect. Now when the force is applied to the edge of a paddle, it swings about that hinge in the expected motion of a pinball paddle.

We encountered difficulties when implementing prior to the creation of our table structure as paddles were completely unconstrained and would rotate freely around the hinge. In the final implementation, the paddle is constrained on its lower bound by the Cannon body it is hinging on, being the paddle rest, and on its upper bound by the table edges. This allows for a very consistent range of motion for the paddle. Without some sort of returning force, there was no way for the paddle to return to its original position, so we imparted a weaker constant force in the opposite direction as the impulse force to return the paddle back to its original position after the paddle was used. One area where we had to be cautious with the Cannon.js physics engine was that, for any force too large or motion too quick, there was the possibility of the bodies in the engine to clip past one another. This was especially prominent with the paddles and the return force, as we wanted to balance having the paddles return back at a satisfying speed such that the player was able to make quick reactions and not having the paddles pass through the paddle rests.

Another possible method of implementing the paddle movement could have been through modifying the paddle body rotation direction on a key press instead of using a hinge and an impulse force. However, we estimated that this would lead to less natural paddle movements due to the way rotating the paddle directly would have a constant rotational velocity rather than having a force like the hinge allowed for.

Terrain Generation:

The terrain is generated chunk-wise, namely that chunks slide towards the user off screen above the table. Each chunk has one pile, where the position of the pile and the height and distribution of the cubes in the piles are randomly generated. The piles are generated using the simplex-noise library and varying the heights of a planar mesh structure. Just before the chunks and piles come into the camera's view, the piles are 'cubified' and the y-heights are sampled. Stacks of cubes at each xz-coordinate are generated up to this sampled y value. Once the cubes are created, the chunk is destroyed and added to the back of the line of chunks and the random pile is regenerated, similar to the approach of the Flying Terrain Generator from 2022.

We looked at using a ring structure which would be visible throughout the pile generation process similar to the Flying Terrain Generator implementation, but we thought we would run into some issues since we have Cannon physics bodies in the scene and the curved movement of the chunks could make them bounce away when they were not supposed to, so we decided on this sliding approach where the chunks would be out of view instead.

Results:

We evaluated our project on a few different metrics. First, in terms of creating a pinball game with dynamic elements to direct the pinball at, we believe we succeeded and had a lot of fun simply hitting the pinball into the piles and watching the cubes which they were comprised of go flying. We found the physics of the paddles and the ball to be satisfying and felt that they adequately represented a real game of pinball. Some aspects of the game do break the desired pinball immersion slightly, such as how spaces in the piles of blocks can be seen and how the paddles interact with the paddle rests slightly strangely when returning, but overall we felt like these were not major issues.

While there are certainly many possible additional elements missing from this game that will be discussed in the Conclusions section, we believe that what we were able to complete was fun, and we felt like that was the primary goal of a game which we succeeded at.

Discussion and Ethical Concerns:

Approach:

Overall, we thought our approach was very reasonable, and the decisions we made to implement both the terrain generation and the pinball movement seemed like the best ones for this project. The only aspect of our approach which could perhaps be improved is the way that the piles are transformed into cubes. Transforming the piles into cubes at a certain sampling rate for every pile creates a lot of cubes, and increasing the sampling rate such that there are no spaces between the cubes like we initially planned proved to particularly slow the program down. Thus, perhaps another way to approach generating these more dense piles of cubes would be to create the cubes after the mesh was collided with by the pinball, then having the physics engine send them appropriately flying away from the impact.

Ethical Concerns:

Two ethical concerns Dynamic Pinball faces are its psychological impact and the reinforcement of destructive tendencies. First, the game's endless nature could certainly be looked at as a game that exploits and leverages psychological reward systems to keep players engaged for periods of time. This risk manifests specifically in younger audiences who might struggle with self-regulation and time management. The ACM guideline to "contribute to society and human well-being" pertains to this issue as developers have a responsibility to design games that promote healthy interactions with their games. Dynamic Pinball could have features that remind the user to take breaks, or the environment gets progressively faster to a point where the user cannot continue.

Second, the game's focus on destruction at its core mechanic could unintentionally normalize destructive actions. The ACM guideline "to avoid harm" emphasizes mitigating the risk of reinforcing negative societal behaviors through digital products. Dynamic Pinball is meant to be a harmless cube destroyer, but younger audiences could be tempted to replicate destructive tendencies in real life. We could include rewards or multipliers for strategic destruction such as you can only destroy certain colored piles while trying to avoid other piles. We could also tailor our theming to emphasize necessary destruction. For example, the game could be nature themed with the far off landscape teeming with life. As the landscape grows nearer, the piles represent less life. These piles must be destroyed to make way for more growth and life closer to the horizon. This would address this concern and allow for a more positive message surrounding actions leading to the destruction of objects.

Conclusions:

We successfully achieved our goal of building a fun and engaging Dynamic Pinball game. The game mechanics and physics came together really well to create an enjoyable player experience. Players can endlessly enjoy demolishing obstacles while trying to keep the ball on the playing surface and not behind the paddles.

The development process was not without its challenges, however. Implementing a live score display using HTML proved to be more complex seeing as we aimed to score using the number of boxes displaced. This meant that we needed to update the score at every time step creating significant performance issues. In the future, we'd want to update the scoring mechanic to allow for users to strive for high scores along with creating a death screen. Finally, we wanted to implement color themes that could be selected in a drop down, perhaps following seasonal color schemes.

This project gave us significant practice utilizing event listeners, camera position, and object-oriented classes to create a dynamic and engaging computer game. Utilizing Cannon.js allowed for interesting physical mechanics that the user could experience first-hand.

Individual Contributions:

- *Vincent Etherton:*
  - I worked on the generative chunks creating dynamically and randomly generated piles of cubes. I created the Cube.ts, Chunk.ts, ChunkManager.ts, and Pile.ts classes, all of which worked together to generate moving piles.
- *Aidan Ward:*
  - I worked on the pinball and paddle movement found in the Pinball.ts and Paddle.ts files as well as creating the table and fine tuning the camera orientation around it in the Table.ts and SeedScene.ts files.

Works Cited:

ACM Guidelines, https://ethics.acm.org/.

CannonJS Documentation, https://schteppe.github.io/cannon.js/.

CannonJS GitHub Page, https://github.com/schteppe/cannon.js.

CannonJS with ThreeJS Tutorial, https://tympanus.net/codrops/2019/12/10/building-a-physics-based-3d-menu-with-cannon-js-and-three-js/.

COS 426 Final Project Typescript and Vite Template, https://github.com/adamfinkelstein/cos426finalproject-vite.

Flying Terrain Generator COS 426 Project (2022), https://github.com/mwcooper/COS426FinalProject.

Lightsaber Dojo COS 426 Project (2022), https://github.com/arcturus3/lightsaber-dojo/blob/master/src/Player.js.

Optional Chaining Documentation, https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Operators/Optional_chaining.

Simplex-Noise Documentation, https://www.npmjs.com/package/simplex-noise.

ThreeJS Documentation, https://threejs.org/docs/index.html#manual/en/introduction/Creating-a-scene.