Introduction

Angry Birds is a popular mobile game which debuted in 2009. In the game, the protagonists are the birds and the antagonists are pigs. The user controls a slingshot, which they use to shoot the birds at the pigs, destroying them and awarding the user with points. The user must acquire a certain number of points to beat the level. Typically, the user is only allowed up to a certain number of birds which they can shoot at the pigs. In the full game, there are different kinds of birds which the user can shoot at pigs. The user normally does not control which birds are in play for a given level. Each bird has a different ability, which can be helpful in certain situations. We have created an interactive game inspired by Angry Birds but with our unique twist. Our game involves shooting Princeton-themed spheres to destroy the other Ivy League-themed spheres by shooting them off the main platform. We combine elements of Three.js and Cannon.js to create a 3D environment where the player interacts with spheres using shooting mechanics and a physics simulation.

Goal

The primary goal of the game is to engage players in an enjoyable experience where they aim, shoot, and strategize to accomplish the task at each level. Gamers and enthusiasts interested in physics-based gaming experiences would find this game fun. It would also interest fellow Princeton students to play a game where the goal is to defeat our rival Ivies. Additionally, developers interested in understanding the integration of Three.js or seeking examples of interactive games would also benefit from exploring the structure and mechanics of this project. Our game may serve as inspiration for developers to add functionalities that we had not originally considered, or to remake the theme into something to their liking.

Previous Work

Many of the examples of implementations of an Angry Birds-style game were in 2D. This makes sense as the original Angry Birds game takes place in 2D space, and this generally offers a more straightforward paradigm, since the developer does not need to worry about the way the added dimension greatly complicates the physics involved. One example of a 3D implementation of Angry Birds we found was at the following GitHub link: https://github.com/paxorus/angry-birds-3d. This project was completed by Prakhar Sahay, a then-student at Brandeis University in Massachusetts. This implementation was written in Threes.Js and uses Physi.Js as the physics engine. Prakhar uses the Ammo.js branch of Physi.Js, as opposed to the Cannon.Js branch. The game begins with the pigs on a platform placed inside structures of rectangular prisms. In the game, the user clicks the screen and holds down the mouse to determine where to shoot a bird, and the power of the shot. The longer the user holds down the mouse, the more powerful the shot will be. It appears as though once the time held

down reaches a certain value, the shot will be at max power upon release. This means there is no need to try to time the release mechanic since shot power will not decrease once it reaches its maximum value. Interestingly, neither the pigs nor the birds are given custom textures within the game. The pigs are green spheres, similar to the color of the pigs in the original Angry Birds game. The color of the bird changes based on the power of an upcoming shot. The bird initially appears on the screen in a color similar to turquoise or light blue. As the player holds down the mouse to increase the shot power, the bird gradually changes color, finishing a bright red similar to that of the birds in the original Angry Birds game. To determine whether the player has won the game, there is a function in the game code that checks if each of the pigs has a negative y value for their position. If this is the case, the pigs are considered to have all fallen off the platform, indicating that the player has won the game. Once the player wins the game, all structures previously on the screen disappear, and the player is met with text on the screen stating “You win!”.

There are a few mechanics in the original Angry Birds game that were not implemented in Prakhar’s version. Firstly, in the original Angry Birds game, the user is given a finite number of Birds to shoot at the pigs in their attempt to knock the pigs off and win the level. If the user finishes their allotment of birds, and they have yet to beat the level, the level ends, and they are considered to have lost. In Prakhar’s version of the game, the user receives an unlimited number of birds to shoot at the pigs and beat the level. This omission serves to remove a key strategic element of the game since birds are no longer a scarce resource that the user must make the most of. In addition to the number of birds, in the original Angry Birds, apart from simply beating a level, the player sought to achieve the highest score possible. In the original Angry Birds game, the user received stars for reaching a certain number of points. Once they reach a particular threshold, they are awarded three stars, the most possible for a given level. The player gained points by either destroying one of the enemy pigs or destroying the structures that surrounded the pigs. In Prakhar’s game, there is no equivalent element of scoring. As mentioned earlier, the player beats the level when all pigs have a y value below zero, but no points are awarded for each pig forced off the platform, and no stars are displayed.

Overall, Prakhar’s game serves as a solid benchmark for what we intend to do with Angry Tigers. The game offers examples of some of the more fundamental aspects of the Angry Birds implementation. It is clear that Prakhar’s approach is successful in terms of using the spheres to represent the pigs, and building the game on top of a seemingly floating platform. These elements of their project give us a good starting point for how we intend to approach the project. At the same time, the game does not completely implement all the features that we would like to include. In addition to the

aforementioned omissions, the game appears to only include one level. While it would be unreasonable to expect the number of levels in our application to rival that of the Angry Birds game, the addition of multiple levels should provide the player with a level of complexity that Prakhar's game does not offer. Additionally, the player in Prakhar's game does not have much of an ability to interact with the game's UI. Apart from playing the level, the player cannot pause or restart the game, apart from restarting the actual webpage. This behavior appears suboptimal, and we intend to build upon this in our implementation of the game. In the context of our project, the omission of these features serves as a positive. This is because it leaves us with room to experiment and build additional complexity onto the game, in particular regarding the scoring mechanic, the way the player interacts with the UI, and the addition of multiple levels.

Approach

The approach undertaken in the provided code is to create a game involving spheres representing different Ivy League school teams (Yale, Columbia, Cornell, Brown, Dartmouth, UPenn), integrating Three.js for 3D rendering and Cannon.js for physics simulations. The "GenericSphere" class establishes a foundational setup for these spheres by defining their basic properties in terms of both visual appearance and physical behavior. From this setup, we can include additional complexity, such as assigning each school a unique texture, size for the sphere, and weight value. These values interact with each other in a way that allows us to vary the difficulty based on the opposing school. This paradigm is similar to that of the provided template for the final project.

This approach should work well in terms of visual representations, the physics simulation, simplicity and reusability, and the ability to modify the game. To elaborate, in terms of visual representation, the use of Three.js for rendering allows for visually appealing representations of the spheres, especially with the addition of school logos as textures. This can make the game more engaging and immersive. For the physics simulation, Cannon.js is used to simulate basic physics properties for the spheres, including mass and collision detection. This should work well for creating interactions among the spheres and with the game environment. Furthermore, using Cannon.js we are ensuring accurate and realistic collision behavior between spheres which contributes to an overall immersive experience for the individual playing our game. We chose Cannon.js since it has a lower learning curve compared to other popular options such as ammo.js. This lower learning curve allows us to get off the ground faster, and spend more time adding functionality instead of learning the ins and outs of the physics engine. In terms of the simplicity and reusability of our approach, leveraging object-oriented programming principles allowed for the reuse of common functionalities while customizing specific attributes for individual school spheres. Specifically, the

GenericSphere class acts as a blueprint, providing a modular structure that allows for the easy creation of different specialized spheres. This approach makes it simple to add more teams or modify existing ones by inheriting properties from the base class.
Lastly, our approach works well when considering integration and modifiability. The code's modular structure, using external libraries like Three.js and Cannon.js, allows for better maintainability and scalability. New features or changes to the game mechanics can be added or modified with relative ease.

Methodology
To implement our game we create a generic sphere class, which serves as the base class that encapsulates the common behavior and properties shared among all the spheres in our game. This class is created in the generalSphere.js file. Next, we created specialized school sphere classes for each Ivy League school. These classes extended the Generic sphere, allowing for the customization of textures and physical properties. This can be seen in the opposingIvies.js file. Each class defines its own values for radius and mass. For instance, YaleBullDogs class has a radius of 2 and a mass of 7, while ColumbiaLions has a radius of 2 and a mass of 7 as seen from lines 5-38 of the file. Inside the constructor of each class, there is a call to load a specific school logo for each Ivy as a texture using Three.TextureLoader. Upon loading the logo texture, this texture gets used as the material for the mesh of the sphere representing their respective teams or schools. We also incorporated a physics engine, specifically through the use of the Cannon.js library for implementing realistic physics interactions between spheres.

For aesthetic enhancements, we textured the platform. The platform is created using box geometry, and the textures are loaded to get the images for each platform’s surface and sides. An array with Three.MeshBasicMaterial objects were created to represent the different sides and textures associated with each side. The top, the main surface of the platform was designed to have the Princeton logo for school spirit. Likewise, maintaining the Princeton theme we chose an orange color for the other sides with the back side of the platform modified to include a special message for the course staff as well. Multiple text elements were also created and styled to enhance aesthetic qualities and to provide key metrics for the gamer to keep track of including the score display, star display, number of lives remaining, number of enemies remaining, time elapsed, etc. To add another visually appealing element to the scene, clouds were created and added to the sky area. A cloud texture was loaded and a sprite material was created using the cloud texture. To make multiple clouds, a loop was created where each cloud’s position was calculated based on the polar coordinates to ensure a circular distribution around the center. An updateCloudsOrientation() function was made in order to traverse through the scenes components and identify the cloud sprites. For each

cloud sprite, a rotation was calculated to make the sprite face the camera as the scene changed or the positioning of the camera changed.

When implementing our game, we considered different approaches to load textures asynchronously, such as directly applying the texture to the mesh or using promises for texture loading. We also examined options to add and fine-tune our physics parameters, including mass, radius, or restitution, to simulate varied physical behaviors. For texture loading, we chose to employ asynchronous texture loading via Three.js TextureLoader to customize the appearance of each sphere based on its respective school logo. For our physics components, we set mass, radius, and other physical properties individually for each school sphere to differentiate their behavior within the simulation. Although the simulation provided accurate collision detection, specific customized interactions or school-specific behaviors weren't extensively explored due to the complexity it would add to the simulation.

To create the levels, we add a URL parameter representing the level. The code then contains a switch statement which adds spheres of different schools on the platform. This implementation makes the process of adding new levels extremely straightforward. When the switch statement checks for level 1, spheres for Brown University and Dartmouth get initialized at specific positions that we calculated on the platform surface. These positions were carefully selected to ensure that there was enough space between each sphere and to avoid collisions with each other. Likewise, the same was done for level 2 and the default level. Yale and Cornell were initialized and carefully mapped onto the platform for level 2 and Columbia and UPenn were positioned for the default level. The newYposition calculation was used to adjust the vertical positioning of the spheres to make sure that they sit properly above the platform’s surface without overlapping.

In each level, the player can receive a maximum of three stars. The first star is awarded for knocking 45% of the enemies off the platform. The second star is awarded after knocking off more than 45% but less than 100% of the enemies, and the third star is rewarded when all the enemies are knocked off the platform.

To navigate between levels, the user can press the “esc” key, which triggers a pop-up menu where they can click the level they want to go to. The user can also press the “L” key (lower case or upper case) to switch between shoot mode and drag mode. In shoot mode, the user can shoot the ball but cannot move the camera. In drag mode, the user can move the camera but cannot shoot the ball. While in drag mode the user can pan the camera, zoom, and move around. While in shoot mode, the camera is automatically

reset to the origin point, looking at the enemies. The camera cannot move in shoot mode.

Game Scene Images

Image 1: Default Level View

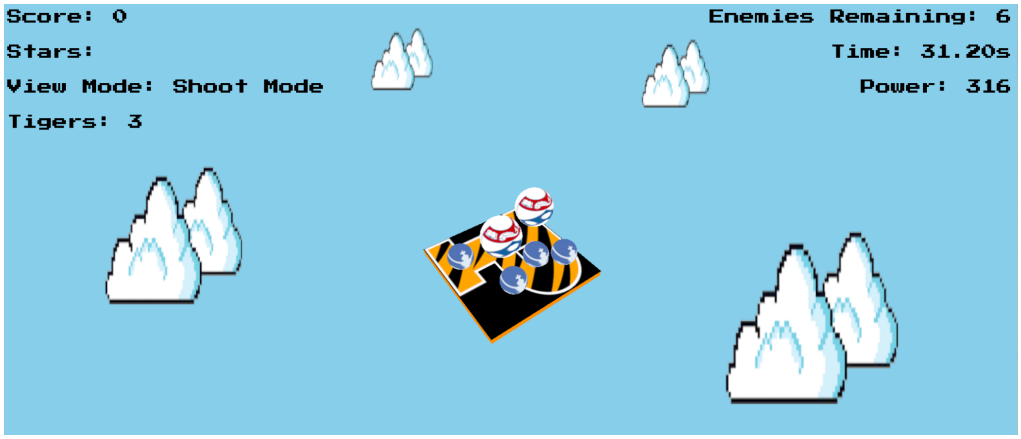

Image 2: Bottom of Platform

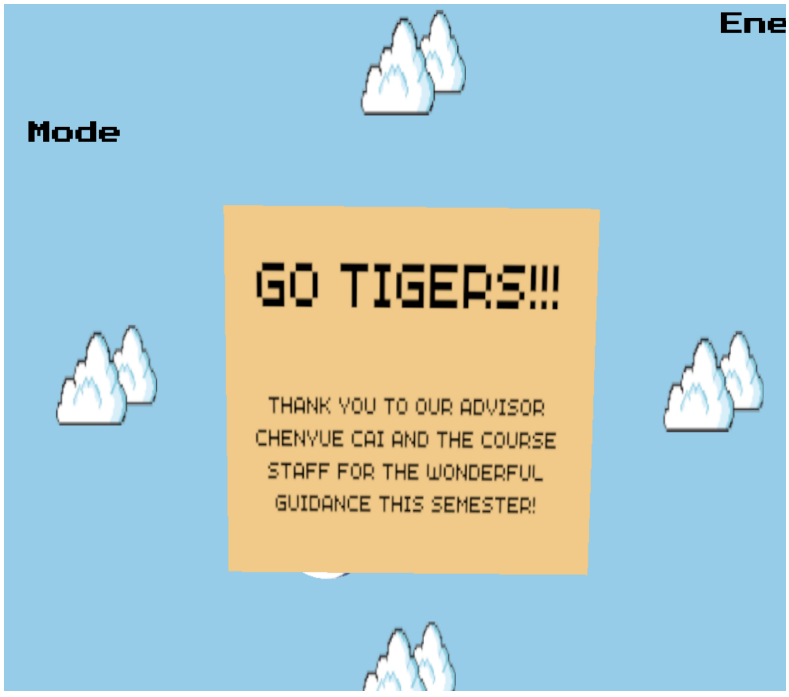

Image 3: Game Over Menu

Image 4: Level 1 View

Image 5: Level 2 View

Results

The success of the project was measured primarily through qualitative assessments of visual representation, interactivity, and collision realism within the simulated environment. Additionally, quantitative measures such as tracking scores, tiger balls remaining to shoot, and time elapsed were employed to gauge game progression and user engagement. To evaluate our game, we allowed users to interact with the scene to observe the distinct representations of each Ivy League school. Feedback was gathered regarding the recognition and differentiation of each university sphere based on the unique appearance of each one. Furthermore, collisions between different university spheres were examined for realism and consistency in adherence to physics principles. This assessment was focused on the behavior of spheres upon impact where the mass and force application were considered. For instance, if a tiger ball (Princeton-themed ball that gets shot to destroy the other balls) hits a smaller and lighter ball, it would be easy to destroy that sphere and push it off the safe platform. However, when the tiger ball hits a ball that is larger and has a bigger mass, it takes more than one hit to affect the opponent's ball. The progression metrics of the game, including score accumulation, remaining spheres(opponent spheres), and time tracking were analyzed to understand user engagement and the game completion rate. From the user engagement and results of the experimental observations, it was evident that users could easily distinguish and identify different Ivy League schools based on the unique textures given to each sphere. In fact, it also built a sense of school spirit to destroy the other spheres (other Ivy Leagues) to win! Likewise, the collision behaviors also successfully reflected the variations in mass and physics properties of each sphere. The score and game time tracking were also accurate. Depending on how many balls were already used, users

were unable to shoot more tiger balls beyond that limit and as a result, saw a pop-up menu telling them if they lost or won and if they would like to retry.

Discussion

We believe the approach we took is solid. We followed a development process that both suited our level of experience and matched our intuition in how to set up the game. In hindsight, it is possible that employing a more object-oriented approach could have simplified our code base and simplified our development process. This could have prevented some headaches of debugging, allowing us to focus more on the more technical aspects of the implementation.

The natural follow-up to our implementation is to add even more attention to detail on the visuals. With the main mechanics ironed out, we could focus on developing precise custom meshes to best represent the object they belong to. Moreover, we could add animations to the enemies similar to those present in the original Angry Birds game.

During the project, our group learned how to set up interactive environments in ThreeJS. We also learned how to implement a physics engine, Cannon JS, and the importance of ensuring the physics world and the ThreeJS world are never out of sync. Additionally, we gained valuable experience testing our game to identify unexpected behavior. The testing process forced us to thoroughly examine our code and find unintended side effects. For example, when determining if the player won the game, we had to account for the case where a player shoots their last shot, but this shot eventually leads to the final enemy falling off the platform. Before testing, our game considered this situation as a loss. An in-depth analysis of our code helped us identify the problematic behavior and subsequently fix it.

Conclusion

This project effectively achieved its primary goal of creating an immersive, interactive angry birds-inspired simulation representing Ivy League schools in a visually appealing and physically realistic environment as our opponents get destroyed by shooting tiger balls at the opponents (Ivy League spheres). Moving forward, the next steps would involve iterating further on user feedback and expanding the gameplay elements to enhance user engagement. For instance, additional levels with additional objects making it harder to destroy the opposing spheres could be added in the future. We can also consider potential scalability and performance optimizations. Overall, this project serves as a foundational step toward creating engaging and interactive educational simulations in a 3D environment for us, and we were able to learn a significant amount along the way.

Contributions
We wish to thank our advisor Chenyue Cai for their helpful guidance throughout the project. We would also like to thank the rest of the course staff in COS 426, including but not limited to Hongyu Wen, Ambri Ma, and Victor Chu.

Note regarding the deployment
The branch of our repo that is deployed to GitHub pages is the gh-pages branch. The branch which contains all the code used to create the game is the main branch.

Works Cited

https://github.com/paxorus/angry-birds-3d

https://schteppe.github.io/cannon.js/

https://threejs.org/