Raccoon Rush

Riya Gandhi, Michelle Huang, Aditya Mehta

Abstract: There are raccoons running around campus! Our game, *Raccoon Rush*, is an infinite-runner game in which a raccoon, controlled by the player, is trying to navigate the many obstacles that may arise on its journey around Princeton's campus. The player uses the arrow keys to dodge obstacles, which include construction signs, construction workers, cones, and electric scooters, all while trying to collect coins. The goal of the game is to get as many coins as possible before hitting an obstacle!

1 INTRODUCTION & GOALS

Inspired by the recent rambunctious raccoon ruckus on campus, *Raccoon Rush* is a third-person infinite-runner game built using Three.js. The player controls the raccoon, which runs on a sidewalk with obstacles one may find on campus – construction signs, construction workers, cones, and zooming electric scooters. Like most classic infinite runner games, such as Minion Rush or Subway Surfers, the primary score comes from picking up some sort of token as you run forward; thus, we chose to adopt a similar approach for our game, where the player must run into the coins in order to increase their score. Running into construction workers will cause this score to halve, while running into all other obstacles will cause the game to end. Once the game ends, the players can easily restart the game and enjoy interacting with the raccoon and other obstacles more. We wanted to create a mindless but addictive game that any player can enjoy when they want to take a break, and were inspired by all the activity happening on campus as inspiration. For this project, we combined the computer graphics knowledge we have learned this semester with our love for classic games from our childhood to develop *Raccoon Rush.*

2 PREVIOUS WORK

When designing our game logic, we mainly thought about two infinite runner games we all grew up playing: Minion Rush and Temple Run. These were both mobile games, with slightly different gameplays:

- In Minion Rush, the player acts as a Minion and guides them through various environments while avoiding obstacles and collecting bananas. This game had a variety of backgrounds and themes inspired by the Despicable Me movie franchise, alongside its soundtrack. Because this is a mobile game, the user swipes left and right to switch lanes, and up and down to avoid obstacles.
- In Subway Surfers, the player acts as a graffiti artist who is trying to escape from the police chasing after them. The environment simulates a set of train tracks, with obstacles such as barricades and moving trains that the player has to avoid. Similar to Minion Rush, the user swipes left and right to switch lanes, and up and down to avoid obstacles.

For *Raccoon Rush,* we took what we liked from both these games to inspire the foundation of our gameplay: an endless running game, in which the player collects some sort of token to increase their score and have the challenge of avoiding obstacles that appear. However, there were some things we did not enjoy about the game, and we learned from that when designing *Racoon Rush*. In particular, for both these games, the player can only move discretely between three lanes in the environment. We wanted to make our game a bit more dynamic, so we chose to allow the player to glide within the bounds of the space freely, with obstacles being spawned both on and

between lanes to increase difficulty. Additionally, Minion Rush was designed in levels, where each level was separate from the others, with a slightly different challenge. We wanted to make a simple game where users could truly get in the "zone" while playing, so we adopted a minimalist approach, where the game could be easily restarted after it ended with just the press of the spacebar, and the user could focus mainly on increasing their high score. As the users become more familiar with the interface, we hope they will continue to want to come back and play our game to improve their score and skills.

3 APPROACH

Once we settled on the idea of an infinite running game themed around current Princeton happenings, we designed an iterative plan for our game. We separated this into three main levels: base case, minimum viable product, and reach goals. For our base case, we wanted to make sure we had the crucial components of the interface: a successfully moving sidewalk, a raccoon that was able to move left and right, and coins that would be spawned randomly. We also wanted to make sure that the raccoon could collide with the coins, which would later be used as the score for our game. For our minimum viable product, we wanted to make sure we had a valid game, with a start and end that a player could play through. For this, we implemented the scoreboard, as well as the start and end menus and the necessary game logic to restart games. Finally, our reach goals included all our ideas for things we could add to make the game more interesting, such as different obstacles, changes to the background, sounds, and more. We worked through these reach goals one by one, prioritizing them by what was doable in the time we had left and what would be most interesting for the player. This approach worked out well for us, and we were able to implement several of the reach goals we had originally thought about.

In terms of tools used, *Racoon Rush* was primarily developed using JavaScript and Three.js. We used the starter code provided to us to set up the camera and renderer, and built upon it to develop our game. We chose this approach because of our familiarity with this library from our assignments in COS426, and because it allows for easy integration with various meshes and perspectives for a 3D game. Elements of the user interface, such as the start menu, coin counter, and game over menu were implemented using HTML and CSS.

4 METHODOLOGY

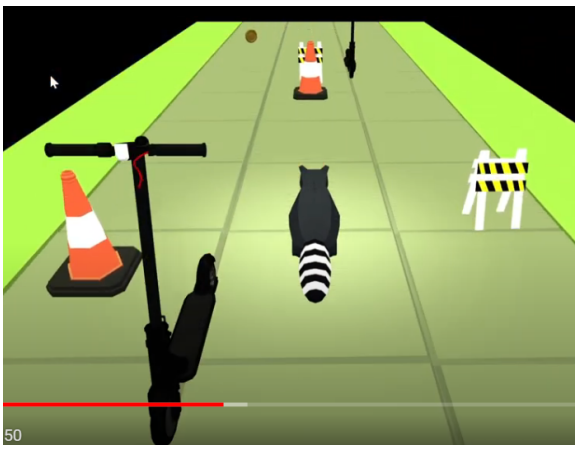
Meshes and Functionalities: The bulk of our game is about the different meshes used and their functionalities to fuel the gameplay. We downloaded all our meshes from open source libraries online and loaded them into our game. Each mesh had its own JS file with its unique properties, and they were all loaded into the main scene based on their needs in the game.

- *Sidewalk:* The sidewalk mesh represents the floor of our game, on which the raccoon and all other obstacles are spawned on. Each row represents a separate instance of the sidewalk mesh. To make the sidewalk endless, we spawn several rows in the scene, and as soon as their position goes beyond the position of the camera, we transport it back to the top of the screen. This allows the raccoon to appear to be continuously moving forward on a sidewalk that stretches on infinitely.
- *Raccoon:* The raccoon is the mesh that the player controls. It spawns in the middle lane of the sidewalk, and is free to glide left or right with the arrow keys to collect coins and avoid obstacles. It is bounded by the grass on either side of the sidewalk. At first, we tried

translating the raccoon by a small fixed amount every time an arrow key was pressed. However, this looked patchy and discontinuous, and so we decided to apply tweening to make the movement more fluid as the raccoon navigates the environment.

- *Coin:* Coins are the tokens that the player aims to collect with the raccoon while playing our game. For dynamic effect, the coins rotate around their vertical axis. We spawn these randomly in the center of any one of the three lanes of the sidewalk. When the raccoon intersects with a coin (more about collisions below), it disappears from the screen and the coin counter increments by 1. We keep track of the coins in a coinList, which we update if a coin is intersected with or leaves the point of view of the game. The coins make the main objective of the game, as the player tries to get the highest score possible by collecting more and more coins, as tracked in the scoreboard on the game interface.
- *Cone + Construction Sign:* These are two of the stationary obstacles we have in our game. Similar to the coins, these are spawned randomly in the center of any one of the three lanes of the sidewalk. We maintain a list of obstacles, hittableList, and update that everytime the obstacles go out of the point of view of the camera angle. If an obstacle is hit, the game ends, displaying a game over screen and the final score.
- *Scooter:* This is another obstacle, but to make it more challenging, they move towards the raccoon at twice the speed of the other two obstacles, which simulates a zooming scooter. Additionally, these are spawned between the lanes of the sidewalk, making them difficult to dodge, particularly if there are any other obstacles nearby. The scooters also go in the same hittableList as the other two obstacles, and if they are hit, the game ends similarly.
- *Worker:* The worker is the last kind of obstacle we have. They are spawned randomly in the center of any of the three lanes of the sidewalk as well. If a worker is hit by the raccoon, the score is halved. This introduces another element to the gameplay, making an obstacle that the user should try to avoid but may run into if they have no other options. We initially wanted to animate the worker to simulate them walking towards the raccoon, but saw some funky behavior where the worker's feet moved in odd ways. We found this humorous, and because it matched our overall theme, we decided to keep this unexpected animation.

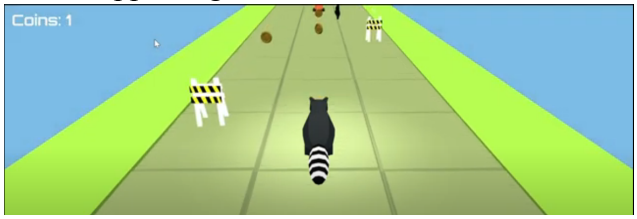
These screenshots capture the various meshes during typical gameplay, with the various obstacles and coins being spawned randomly in the locations mentioned above.



Collisions: Most of the meshes described above required logic to check and process collisions with the raccoon. Thus, we created an overarching HittableObject class that moves all obstacles at the same rate as the sidewalk, and handles collisions. Then, each of the meshes (coin, signs, scooter, worker, and cone) inherit this class to be included in the collision logic.
To model collisions, we assign each object a bounding box and add all collidable objects to the lists coinList (for coins), hittableList (for scooters, signs, and cones), and workerList (workers) in the scene. At each frame update, we check if there are any collisions between the raccoon and any of these objects in these lists by checking the intersection of the raccoon's bounding box with the other object's bounding box. If the two intersect, then a collision is detected. Then, depending on the obstacle/object. For coins, we handle the collision either by incrementing the coin counter and removing the coin from the scene, allowing gameplay to continue. For workers, we half the score and remove the worker from the scene. For the remaining obstacles, we change to the game over state and stop any movement on the screen.

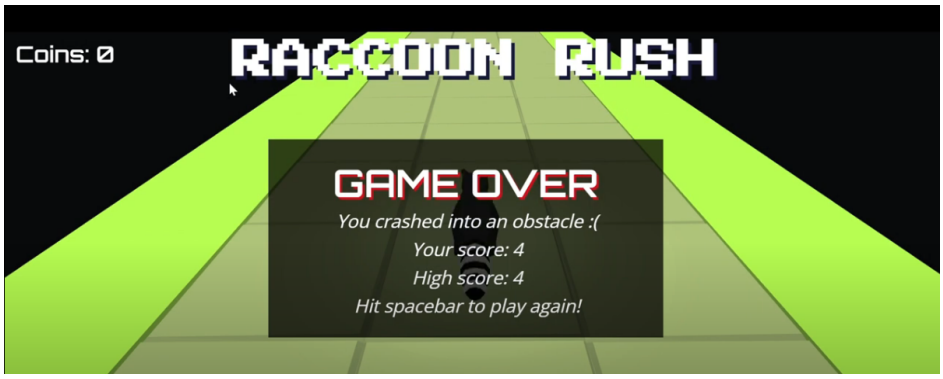
Collisions provided an interesting and frustrating challenge as we were developing our game. Initially, we were manually moving the bounding boxes at the same rate as the objects they were on, but soon adopted methods from the Three.js library to make this easier. We also ran into many issues where slight changes in the placements of obstacles would affect the bounding boxes, and figuring out why this was occurring involved a lot of debugging. Sometimes, the raccoon would visually go through objects when collisions should have been happening. We had to perfect this for each obstacle, and make sure the game was behaving as expected when two objects intersected.

Scenery: To maintain a dynamic background, we gradually go from day to night and back during gameplay. Below, we see the daytime on the left and the nighttime on the right. A more clear view on how this change is gradually made to simulate time can be seen in our video demo. The scenery of our game consists of the sidewalk being surrounded by grass on either side, with all the gameplay occurring on the sidewalk as described above. We also edited the lighting to create a subtle spotlight on the raccoon when it first spawns, seen in the picture on the left. It also created appealing reflective effects on the sidewalk, making the scenery a bit more dynamic.



Start Screen: When the player first enters our game, they see a stationary sidewalk, with an instructions menu in the center of the screen that lists out basic rules and how to play the game. We also see the title of our game and the coin counter initialized at 0. We used JavaScript code to build HTML elements, and CSS to stylize them with different fonts that matched our theme.

End Screen: When the player hits a cone, construction sign, or scooter, they are met with the end screen. The sidewalk stops moving, the coin counter is reset to 0, and the raccoon is reset to the middle. In the end screen, the player sees the score they achieved in their most recent run, as well as their high score. This encourages the user to quickly press the spacebar to play again, hopefully working to improve their high score.



Sounds: We implement sounds on several events in the game.

- *During gameplay*: While the game is in play, we have an instrumental version of Eye of the Tiger by Survivor playing as the soundtrack of the game. This stops when the game ends, and restarts when the game is restarted.
- *When an obstacle that ends the game is hit:* When the player hits a cone, construction sign, or scooter, we play a loud 'smack' sound.
- *When the worker is hit:* When the worker is hit, we play a short sound of coins jingling, symbolizing the loss of coins being "robbed" from you, as the coin score is halved every time this collision happens.

5 RESULTS

We measured success in terms of achieving all the functionalities of the game that we planned for. Our game is fully functional, with polished and realistic graphics and accessible user interfaces. Experimentally, we evaluated our game through extensive debugging and testing, reaching all possible edge cases and scenarios that users could encounter in our game. For example, we collided into every type of obstacle, exercised the full range of movement of the raccoon, and made sure all the sound effects executed at the appropriate and intended time. Through our testing, we made sure that the game behaves as we expected it to. We also tested

how users could interact with the interface by iterating through possible combinations of keystrokes during the game as well as when starting and ending the game.

Finally, as this project is intended to be a user-friendly, entertaining game, we had friends try our game, given minimal context or instructions in order to assess how a real user would adapt and interact with our game. We had iterations of feedback from our beta-testers that are now implemented fully in this final version.

6 DISCUSSION AND CONCLUSION

Through this project, we learned to utilize graphics and principles of design and physics to implement a realistic game. Our approach was to design a simple, easy-to-play game reminiscent of Princeton culture, and we were able to execute our vision of the game as intended. Though the controls are simple, the game feels dynamic through the rapid pace of obstacles, day and night scene changes, and audio effects. Alternative infinite-runner games could introduce more complex animations and decorations, but chose to keep the design simple to focus on interacting with the actual objectives of the game.

Future work would focus on implementing more features and variations in the game, such as adding sound effects when collecting coins, varying the speed of the game according to player-selected difficulty levels at the start of the game, and creating more interactions with obstacles such as jumping over or sliding under. We could also improve the aesthetics of the game by adding more details or effects that would more closely resemble an actual runner in the scene.

Overall, we enjoyed exercising our creativity and technical skills in this project, with a newfound appreciation for game development and the work that goes into every tiny and oftentimes overlooked detail of a game. We were able to implement all of our goals for this game, and along the way, leverage Princeton humor and campus phenomena to personalize this game.

7 CONTRIBUTIONS

Riya Gandhi: Coded start and ending menu with designing interfaces, raccoon movement and bounds, interface with CSS, added sound effects, wrote paper

Michelle Huang: Coded coin dynamics and counter, obstacles and customizations, interface with CSS, added soundtrack, wrote paper

Aditya Mehta: Coded infinite-runner path movement and design, collisions with obstacles, added sound effects

8 WORKS CITED

Project template provided by COS426 and Reilly Bove ‘20.

Inspiration:

- Minion Rush/Subway Surfers

- HTML/CSS: https://github.com/justin-bi/3D-Breakout

Sounds:
- Soundtrack: https://x-minus.pro/track/50746/eye-of-the-tiger
- Sound effects: https://pixabay.com/

Meshes:
- Poly Pizza

Libraries:
- Three.js