COS426: COMPUTER GRAPHICS
FINAL PROJECT WRITEUP: Rabid Raccoons

Dylan Epstein-Gross, Saarthak Chaturvedi, Suhani Balachandran

I. ABSTRACT.
"Rabid Raccoons" is a third-person infinite enemy generator game built using Three.js. The player controls the Campus Fox maneuvering through a forest, as raccoons endlessly spawn out of the trees and come towards the player. The goal of the game is to stay alive as long as possible, but if a raccoon touches the player, the game is over. This tests the player's ability to multitask and act quickly.

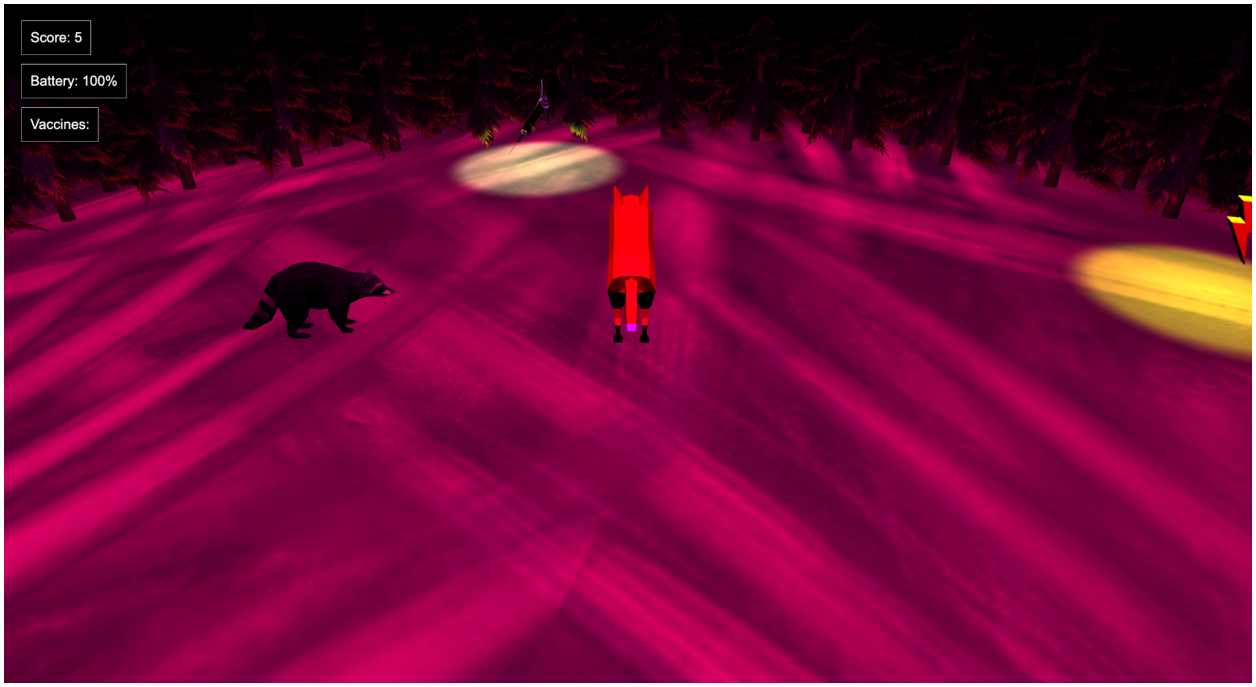

Figure 1: Playing Rabid Raccoons!

II. INTRODUCTION.
Goal. When considering what game to make, there were several characteristics we considered and prioritized, all detailed below.

Concept. We wanted to make the concept of the game topical and relevant to Princeton students, while also making the story simple so that the concept and directions itself were not a barrier of entry to gameplay. We thus chose to have the Campus Fox, a University legend, play against rabid raccoons, a phenomenon recent in campus news. Anyone who has learned of our concept has at least smiled at the mention of these famous animals, thus proving that this was a good approach.

Three-dimensional. We wanted to make our game three-dimensional because we all felt that the 3D-focused learnings from the class were most personally compelling, and we wanted to give ourselves the extra challenge of navigating 3D interactions.

Infinite. As opposed to having designated levels, and thus a clear end to the game, we wanted to create a game that could theoretically go on forever. This ensures that our game has endless replay value, as opposed to a game which cannot be played again with the same amount of fun once you have beaten the levels.

Enemy generation. Once deciding upon having an infinitely generating instead of level-based game, we weighed the options of doing an infinite runner or an infinite enemy generator. We decided on an enemy generator to really feature the raccoons and thus stay the most true to our concept.

Interesting gameplay. We wanted to make sure there were enough different features to keep it engaging through the endless nature of the game. We thus implemented a weapon to ensure that the player can be more active in the game than just running away from the enemy; increasing risk as the game goes on through the degradation of the weapon and faster enemy spawn rate; powerups that can reverse the degradation of the weapon; and the ability to get and improve on a high score. All of this ensures that our endless game truly does have replay value and will be exciting to play. The details of the implementation of all of these described features can be found in Section III, Methodology.

Previous work. At our first stages of ideation, we thought about making a conversion of a well-known, nostalgic two-dimensional game into 3D, and the idea that first came to our mind was Pacman. While the look is obviously different, many of the gameplay ideas were influenced by that classic game. Moving on the terrain is our 3D analog of Pacman's maze, and we also offer powerups to collect. The excitement and fear created by the constantly approaching and regenerating enemies is reflected in our game too, as is the idea of lovable cartoonish character design. The concept of Pacman is simple, but lends itself to such endless fun that it has found a place in the classic games pantheon. It thus offered us the perfect framework to ensure that we were making the best simple, fun, endless 3D game that we could. That said, we also wanted to take an angle that would distinguish us from other bright and exciting game entries, and we found that the Three.JS lighting and camera system were especially conducive to a more horror-like experience. Our creation was also inspired by classic "spooky" games like Luigi's Mansion, incorporating scary aspects to increase the engagement of players.

Approach. We started with the starter code repository provided by the COS426 teaching staff, which, as stated, uses Three.js and 3D object models to create a basic scene and JavaScript event handlers for interacting with the scene. We added to this using our own logic, other Three.js components, and open source character models. Based on our broad goals, our MVP game version was one where the enemies endlessly generated and the fox had to avoid them and disable them. Our stretch goals included obstacles, powerups, a second type of enemy, UI elements, animations, and sound.

III. METHODOLOGY.

What was implemented, and how.

User controls. The player is able to interact with the scene using key presses that activate event handlers. All of the controls are described on the beginning start screen: WASD to move up, left, down, and right respectively, K to start the game, spacebar to use the flashlight weapon, and L to use the vaccine syringe weapon. Our first implementation moved the fox in world coordinates only, however this sometimes created an awkward movement where W did not necessarily mean forward for the player (such as when the fox was facing a different direction). We thus used linear transformations to change the movement to be based on the camera coordinates instead, so that WASD means away, to the left, toward, and to the right of the camera. Our first approach to movement also resulted in overly "discrete" motions (only moving on vertical and horizontal axes in accordance with the keys), which seemed unrealistic; thus, we used some geometry to incorporate a more gradual rotation into the fox's turning, resulting in a smoother experience.

Figure 2: Fox Model

Camera. The camera is a Three.js perspective camera. We wanted to create a third-person game but still replicate the challenge the player would have in a first-person game of not being able to anticipate enemies spawning from behind. We thus have a camera whose position is a based on the position of the fox, where its x and z components are based on trigonometric functions of the angle of the fox's x and y coordinates. The camera always looks at the fox's position. By some tuning, this accomplishes a camera that is held away from the fox but still has a limited viewpoint, and follows the fox's "forward" focus. We tinkered with the angle and distance of the camera to create a spooky perspective where the player can still see enemies coming from behind them.

Scene. The scene was developed using a number of 3D models. The land is actually a cube with an open-source texture applied to it. Just at the perimeter, there are four layers of pine trees (created using an open-source 3D model) to create the illusion of a densely-populated forest beyond the perimeter of the gameplay area, and to provide cover for where the enemies spawn. The background was made black and the lighting was just a spot light, basically shadowing all beyond the perimeter to give it a natural, eerie night-time look that extends into seemingly infinite woods.
This scene was arrived upon after an extensive amount of changes. We started with a vastly different terrain, an irregularly shaped island 3D model that we scaled up to give the illusion of a never-ending forest but still make it clear that the land was irregular and thus add a bit of a realistic approach. We then manually placed one layer of trees around this irregular perimeter, and then tried to programmatically generate more layers of trees in different locations based on this first layer. All of this did not end up looking how we wanted it to, created too small of a gameplay area because of the irregular land shape, made managing collisions between the trees and fox difficult, depended too highly on manual inspection, and was not conducive to changing parameters of the environment, such as the size of the land. We thus switched back to using the given square based land. Then, we tried randomly generating different types of trees in different spots across the land, but it just looked far too sparse for the purpose and was prone to odd irregularities with trees overlapping. Finally, we ended up using one type of tree and generating four rings of them to circumscribe the play area, pushing the trees out to the edges of the camera focus to create the endless forest illusion that we desired. Next, we tried adding a bed of 3D grass to make it look more like a forest, but using enough grass that the entire land was covered in a realistic-looking way severely lessened the frame rate and made the game very slow. We thus had to compromise on our design wishes and instead made do with a flat texture. Since we weren't able to do natural-looking elements like grass, we decided to go in the other direction and play with the game feel and the eerieness, opting for a scary almost-blood-toned texture for the game arena. There is much more we wanted to do with the scene, especially adding randomness in where the scene elements are located, but they were less worth our focus than the actual mechanisms of gameplay.

Entity spawning. As the game progresses, two types of enemies are repeatedly spawned in – standard and "big" raccoons. The enemies are spawned in randomly chosen locations along the perimeter of the circular clearing, chosen such that they are not initially close to the player. The spawn rates were initially kept constant, but later we decided to make the game's difficulty increase over time by decreasing the gap between spawns as the game progresses. This creates a more frenetic feeling as enemies spawn faster and faster. That said, we had to cap the spawn rate to prevent over 20 raccoons from being present at a time, as this would too quickly overwhelm the player and slow the frame rate to a crawl. Spawning for powerups was similar, with the exception that they can spawn randomly anywhere within the circle and that their spawn rate decreases over time rather than increasing.

Enemy behavior. At first, we simply programmed the raccoons to move directly toward the player with constant speed every timestep. However, we found that this behavior was too smooth and did not feel "rabid" enough, so we adjusted the tracking motion to add some randomness. At random intervals, a random vector is chosen to be consistently added toward the motion, thus causing the raccoons to occasionally jitter and head off course before correcting toward the player. Additionally, the speed of the raccoons can change by small random increments at random times, increasing the excitement as one raccoon may lunge forward or fall back. Contrary to the standard raccoons, the big raccoon has a slower movement speed and does not move randomly, making it more intimidating and difficult to avoid.

Figure 3: Raccoon Model

Flashlight weapon. Creating a light-based weapon was daunting, but it was thankfully possible due to the impressive Three.JS library. We were able to create a SpotLight that

followed the player and always pointed ahead of it using some simple geometry; then, when the spacebar is pressed, the intensity of the light becomes positive. Any raccoons within the light cone will back away from the player, losing HP until they are “defeated” – which causes them to turn around and run into the forest. However, to increase the challenge, we designed the flashlight’s radius and angle to decrease while being used, making it more difficult to block the raccoons. The flashlight went through several iterations, as it often caused unrealistic movement of the raccoons; we ultimately found it felt smoothest for attacked raccoons to keep moving backward for a little longer than when in the light cone, which was difficult to program yet greatly improves the feel of the weapon.

Battery powerup. The battery percentage of the player’s weapon, the flashlight, decreases as time goes on. Collecting the bolts enables the player to have a boost in their battery percentage so that they can continue to live and play. The bolt starts with a free downloaded 3D model, which randomly spawns around the screen with a spotlight over it. If the distance between the player’s position and the position of the bolt is less than the determined collision radius, then the bolt disappears and the battery increases (with a clamp at 1). An array of bolts is maintained in SeedScene.js, and added to dynamically with a function we built to calculate a random position for the bolt to spawn that is also a function of the player’s current position. Within each bolt we also compute the current lifetime, reducing the intensity of the spotlight until eventually despawning the battery if the player does not reach it.

Vaccine powerup. When playing with just one attack, we noticed that it was difficult to scare off raccoons behind you, and also that the big raccoon was too easy to push back – this motivated us to add a second powerup. The syringe powerup represents a rabies vaccine and, upon its use, repels all raccoons near you, including the bigger raccoon. Similar to the lightning bolt, it also begins with an open source downloaded 3D model, which randomly spawns around the screen based on the function we built that introduces randomness but is also based on the player’s current position. Collisions are handled similarly. However, the player interacts with the syringes differently. They are not used immediately. Instead, they go to inventory, where the number the player has collected (stored in a simple counter variable in the player’s class) is displayed in our generated UI element and continuously updated using JS. This powerup also involves a keypress, handled with an event handler. Upon pressing the key L, this activates the second type of attack from the player, handled in a method we built in the player class. This sends a light blast of a decided upon radius, makes the enemies turn back away, and decreases the number of syringes.

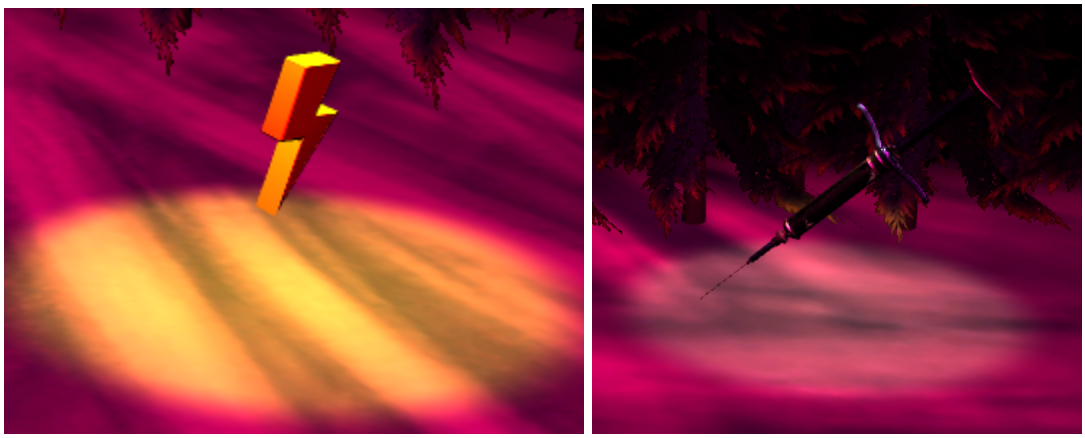

Figure 4: Powerups

UI elements. We made the UI elements by creating functions in the main `app.js` file that would create and modify HTML elements within JS based on occurring events. We used this same approach to create the start and end screens, that activate when you start the game or when the game is over, and which give away to the actual game when you press K to start or R to refresh, respectively. We also have small elements on the side of the screen that show the score based on the number of seconds survived, the current battery percentage, and the number of vaccines in the player's current inventory. These elements are also initialized in functions in `app.js` but use values in the scene to update their display based on the user's progress. We made sure to keep track of the user's current score and high score to make the experience more replayable, as user's compete to survive for longer!

Figure 5: Start (left) and game over (right) screens

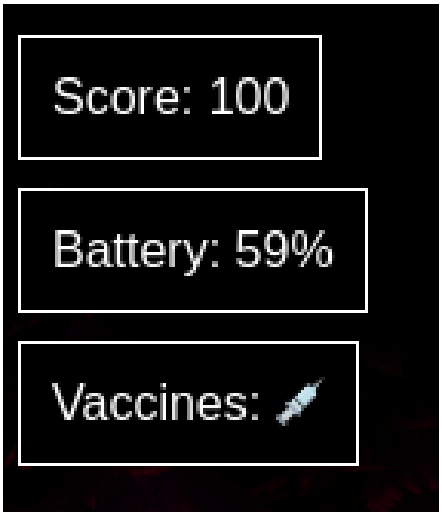

Figure 6: UI Elements

What wasn't implemented, and why.

Obstacles. One of our stretch goals was to implement an obstacle like an animal trap. The idea was it could introduce more strategy into the game. The animal trap would be harmful to the fox, and thus another thing the player had to think about avoiding in addition to the enemies. However, it would also be harmful to the small raccoons, so another way the player could disarm them would be to lead the raccoons to the trap, while being careful to avoid it themselves. But, it would not be able to disarm the large raccoon. There were a number of reasons why we were not able to implement this. While the spawning logic would be similar to the powerups, it would be difficult to navigate their different behavior with the raccoons. Currently the only collision we have implemented for raccoons is with the player itself, which does not continuously generate, unlike the traps. This would add another layer of complexity that we were unable to get to in the current version, but would love to add if we were to work on this in the future.

Animations. Some of the 3D object models we downloaded came with built-in animations that we wanted to try to include, such as a walking animation to make the movement of the characters more natural. However, even after referencing past successful projects, countless pages of documentation, and debugging forums, and then trying to work on it with a COS426 graduate TA for hours, we were unable to have the animations appear. This is definitely something we were interested in and spent a lot of time trying to accomplish, but ultimately had to abandon for the sake of other elements.

Sound. Similarly to the animations, after referencing past projects, documentation, forums, and trying to work through it with a TA, this did not come through before we felt we should divert focus to something else.

Environment: As mentioned earlier, we tried to implement a more complex environment – with more obstacles like rocks and trees, or procedural grass – but found that we were very limited by the decreasing frame rate. Perhaps it was just our machines, but any more than a few trees and our browser would crash trying to render the game. This taught us an important lesson about environmental complexity, so we decided to keep many aspects of the game as minimalist as possible.

Better Enemy AI: We were also surprised to find how difficult it was to implement a more complex enemy AI. While we initially had hoped for raccoons that would repel each other, sneak around the player, etc., we found that many of our simple solutions resulted in jittery and unrealistic looking movements. As a result, we had to settle for tracking with some randomness, though in the future we would hope to look further into the literature for novel enemy behaviors.

IV. RESULTS.

Results were gleaned from constant testing of the game, particularly with the help of the UI elements to fix user experience bugs. For example, we saw that we were able to reach a high score of 300 (thus making the game a bit too easy) because of the constant availability of the vaccine powerup. Thus, we adjusted its spawn rate. We also had other students or friends take a look to improve aesthetics or movement concerns.

V. CONCLUSION.

Efficacy. We believe we achieved our goal quite effectively, as we were able to implement all MVP goals and half of our stretch goals. The game has a clean, pretty look, is exciting to play, and incorporates a relatable and fun concept.

Next steps. Our next steps would definitely be to implement the stretch goal features that we were not yet able to get working: the animal trap obstacles, animations, sound, a more complex environment, and better enemy behavior. We'd also like to include a third type of enemy, and more types of weapons and powerups. From a UI perspective, also adding the ability to pause the game would be helpful, as well as a leaderboard.

VI. CONTRIBUTIONS.

We all worked together on every step of the process. After achieving the MVP together, we then split our focus to Dylan working on powerups, Saarthak working on the UI elements, and Suhani working on the scene and animations/sound.

VII. WORKS CITED.

3D object models.

- Raccoon: https://skfb.ly/otsOA
- Fox: https://skfb.ly/6GIrP
- Syringe: https://skfb.ly/6WLNp
- Lightning bolt: https://skfb.ly/oOq7W
- Tree: https://skfb.ly/o7tMy

References.

- Code skeleton created by Reilly Bova ‘20
- Three.js documentation