

# COS426 Final Project Written Report:

## Triathlon Mania

Palzom Lhamo, Lauren Pak, Ruoming Shen

### Abstract

This project involves the creation of *Triathlon Mania*, a Three.js endless runner game in which the player competes in an infinitely long triathlon and must avoid hitting obstacles in order to stay alive. We aim to encourage user engagement with this game by creating an infinitely moving terrain rotating between running, swimming, and biking paths; randomized obstacles and rewards on the player's path, which change based on the current terrain; and an animated player—whose appearance and movement change based on the current terrain. User experience tests with *Triathlon Mania* suggest that the game is enjoyable, fun, and visually pleasing. Our approach of creating an endless runner game with changing settings, character movements, and obstacles was successful.

### Introduction

The goal of this project is to create an infinite-runner style game paying tribute to existing endless runner games like *Subway Surfers*, *Temple Run*, and *Crossy Road*. These popular games have a common goal: to stay alive for as long as possible by avoiding obstacles. In this project, we seek to emulate this style of game and add innovative features to increase player engagement. Thus, we present *Triathlon Mania*, an endless runner game in which the player competes in an infinite triathlon and must avoid terrain-dependent obstacles in order to survive. Drawing inspiration from the aforementioned endless runner games as well as an open-source Three.js game titled *Boxy Run*, *Triathlon Mania* keeps players hooked with randomized obstacles and rewards, interesting scenery, and a relatively fast game pace.

In the vast majority of existing endless runner games, the scene, player's avatar, and obstacle types remain constant for a given run. Therefore, this project attempts to maintain players' interest during a run by dynamically rotating through different terrains and characters. The distinctive triathlon theme of this game opened several avenues for creative game design, especially in regard to character, animation, and scenery generation. Depending on the current phase of the triathlon, the player navigates a different terrain (field, ocean, mountains) as a different character (runner, swimmer, biker). Our hope is that *Triathlon Mania* will provide an engaging and fun experience for anyone looking to relax.

Our approach combines the successful aspects of *Boxy Run* and *Subway Surfers*, particularly their infinite terrain and simple controls, while also offering a twist to traditional endless runner games with our triathlon theme and intricate elements such as changing terrains and player movements. The premise of *Triathlon Mania* is as follows: the triathlon participant (i.e., the player) has three lives and must run, swim, and bike for as long as possible. Every time the player hits an obstacle, they lose a life. If the player has less than three lives, they can consume a reward on their path to regain a life.

### Methodology

The MVP for *Triathlon Mania* involves three main components: an infinite moving terrain rotating between running, swimming, and biking paths; randomized obstacles and rewards on the player's path, which change based on the current terrain; and an animated player—whose appearance and movement change based on the current terrain—with two degrees of freedom. Implementing these essential components presented several design and technical challenges, which are outlined below.

## 1) Moving Terrain

To simulate the three tasks in a triathlon, we create a racing path using a Three.js BoxGeometry that is positioned slightly above the remainder of the terrain, which is rendered as a PlaneGeometry. In every timestep, both the racing path and surrounding terrain move toward the player according to the configured game speed. When the path's z-position reaches the player, the path and terrain's z-positions are decreased to simulate an extended path. After the current path's position has been reset three times, the path and terrain corresponding to the next portion of the race are made visible and positioned at the end of the current path.

We adopt this approach of reusing geometries and meshes—as opposed to creating new objects upon a terrain switch—in hopes of improving the runtime performance of our game. By simply adjusting the position or toggling the visibility of a singular object, we minimize the frequency of generating new objects, which can be computationally expensive. Another optimization we implement is shortening the length of the path and periodically adjusting its z position backwards to mimic a longer path. This optimization reduces the memory usage of our system.

To keep track of the terrain state and any imminent terrain switches, we define a new class named TerrainController. This class stores the number of “laps” (i.e., the number of times the current path's position has been reset) completed for the current terrain, as well as the current terrain, which is represented by the TerrainPhase enum. The TerrainController's current state dictates the visibility of all objects in the scene as well as which athlete the player currently is. The TerrainController is also used to indicate when to make the next terrain visible; in specific, the next terrain is made visible when the current terrain has reached two laps. This algorithm ensures a smooth, gapless transition between terrains.

## 2) Terrain Graphics

Each terrain is created using a different computer graphics algorithm, creating a unique experience for each phase of the infinite triathlon (see Appendix A). During the running phase, grass is simulated with a fragmented-like effect. This effect is created by segmenting the grass plane, assigning a random shade of green of each vertex in the plane, and interpolating the vertex colors to color the pixels on each plane face. To create animated ocean waves for the swimming phase, the ocean plane is similarly segmented into triangular faces, and the x and y-coordinates of each vertex continuously change based on a sinusoidal function. Finally, the mountains visible during the biking phase are generated by applying Perlin noise to a segmented plane. Utilizing the simplexNoise library, we create a noisy heightmap that determines the y position of each vertex in the plane, and we subsequently assign a color to each vertex depending on its y coordinate value.

## 3) Obstacles and Rewards

The type of obstacle in the scene varies by terrain: deer are in the running phase, sharks are in the swimming phase, and low-flying birds are in the biking phase. Using bounding boxes, we check for collisions between the player and all current obstacles in the scene. A collision is detected when the bounding box of the player—obtained via `Box3().setFromObject()`—intersects with the bounding box of an obstacle. To reduce unnecessary collision checks, this bounding box calculation is not performed for obstacles that are currently not visible in the scene and for obstacles further than a fixed distance threshold from the player.

As with all objects in the game, the obstacles move toward the player, with its z coordinate value increasing by the game speed in each timestep. When an obstacle becomes out of sight (i.e., its z position

is greater than the camera's z position), it is moved back behind the scene's fog by a randomized z amount ranging from 200 to 250. Its x coordinate is also randomly reset to one of three configured x values, each of which representing a lane in the path.

The deer are static, and the player can circumvent by moving to a free lane or jumping over the deer. In contrast, the sharks move in a zig-zag direction toward the player—this movement is implemented by updating the sharks' x and z coordinates at every timestep (in addition to the z-coordinate update performed on every object in the scene). These two coordinates are updated in the direction of  $\pi/6$  or  $-\pi/6$  depending on the current orientation of the shark. The player can circumvent the sharks by either moving to a free lane or diving under the shark. To create the illusion of the birds flying through the mountains, the birds' x coordinates are updated at every timestep. The player can circumvent the birds by switching lanes or briefly jumping over their bike and over a bird.

Like obstacles, the type of reward in the scene also varies by terrain: acorns are in the running phase, treasure chests are in the swimming phase, and lightning bolts are in the biking phase. The player's collisions with rewards are detected using the same bounding box algorithm as obstacles. Upon colliding with a reward, the player only collects the reward if they presently have less than three lives. If the player collects the reward, they gain a life, and the reward's visibility is toggled to false. Otherwise, the reward is not collected, and it remains visible despite the collision.

In the running and biking phases, collisions between the obstacles and rewards are also detected in order to simulate the obstacle animal “consuming” the reward. If a given reward collides with an obstacle before it collides with the player, it is made invisible and reset to a further position, returning to the player's vicinity in a much later timestep. Obstacle-reward collisions are not detected in the swimming phase because the diagonal movement of the sharks and the vertical bobbing movement of the treasure chests make this type of collision highly likely.

To incentivize player risk taking, we add special properties to some of the rewards. These properties are detailed in the Discussion section.

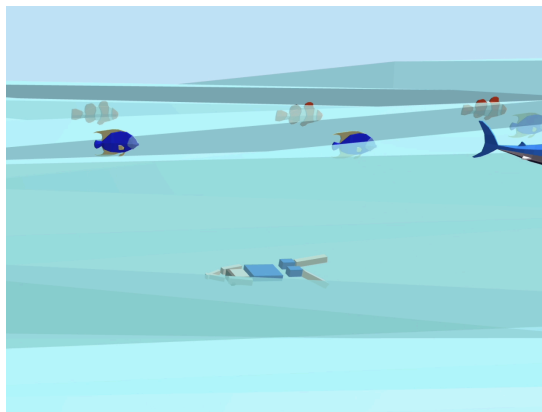
#### 4) Terrain-dependent Player Movements and Appearance

Each terrain comes with a change in the player's movements and appearance. In the running terrain the character is upright in a jogging position, and the movements are up, left, or right.



Character jumping in the running terrain (purple shirt, blue pants)

In the swimming terrain the character is in a swimming position and the movements are down, left and right. In the swimming terrain pressing down allows the character to dive into the water to avoid sharks.



Character diving below the water's surface

In the mountain terrain the character has the same controls as the running: up, left, and right. Pressing up allows the character to jump off of its bike to avoid the birds.



Character jumping in the mountain terrain

To first create the character, we use Three.js's mesh boxes, cylinders, and group containers to create the characters' limbs and define its initial position in the `init()` function. To animate the character's running movements, every limb's x-rotation is assigned to a number calculated by the sinusoidal function. Through its parameters—specifically, the frequency (the number of limb swings per second), the minimum and maximum rotation angle (how far the limbs will move back and forth), and the phase (for swapping limbs), the sinusoidal function returns an infinite looping animation of smooth running, swimming, and biking movements as long as the game is not paused.

With every switching terrain, the character seamlessly switches accordingly. To implement this, we keep track of `TerrainController`'s `characterPhase` and, if it is not equal to the character's `currentPhase`, the current character is removed from the screen (with its current x-position stored as a temporary value for the next) and popped out of the `updateList` list keeping track of the character's phase. The new character (a new `Runner` object, `Swimmer` object, or `Biker` object) is initialized based on `TerrainPhase` and assigned the former character's x-position for a smooth transition, adding it to `updateList` and updating `currentPhase` accordingly.

If the right arrow or left arrow are pressed, the character switches right and left, respectively. If the character is `SwitchingLeft` (which is called upon the `ArrowLeft` action), the character's position is decremented by 2.3, a lane amount. Vice versa, if the character is `SwitchingRight` (upon the `ArrowRight` action), the character's position is incremented by 2.3.

## 5) Additional Features

### Scenery

The addition of scenery to each of our terrains helped make the game more visually appealing and ultimately help maintain player engagement. Some of the objects we created are more triathlon themed like the spectators cheering that are in the running terrain, while other objects are more terrain focused like the dolphins and fish in the ocean and the greenery in the mountain terrain. We decided to animate objects like the fish in the ocean and the spectators on the sidelines to add more action to the game and make it more aesthetically pleasing.

We employed multiple techniques for animating the objects, including looping the pre-set animations for some 3D models and creating our own animations using sinusoidal functions. To animate the sea creatures (and the birds during the biking phase), we used the Three.js AnimationMixer and AnimationAction classes. When loading the GLB file for a sea creature object, we extracted the animation clip contained in the file and added an AnimationMixer containing the clip to the object's state. This mixer is updated on every timestep to continuously render the animation.

To simulate movements that the pre-set animation clips do not provide, we manually adjusted some objects' positions and rotations at each timestep. For example, to simulate dolphins leaping out of the water, the y and z positions of the dolphins fluctuate based on different sinusoidal functions, which take into account the frequency and magnitude of the leaps. Similarly, to animate the cheering spectators during the running phase, we adjusted each individual limb's position and rotation according to sinusoidal functions.

For the background, we incorporated a temporal gradient so that the current color morphs into the next color smoothly under a color fade duration. Keeping track of the elapsed time using Data.now, we computed the interpolation factor and, taking the current color and the target color it is morphing into, we used Three.js's lerpColors() function to linearly interpolate from the start color to the target color using the interpolation factor.

### Sound Effects

The sound effects were implemented using the built in JavaScript Audio class. First we created individual audio objects for each sound effect. The initialization and loading of these Audio objects occur at the creation of the scene and/or character that the sound effect is tied to in order to prevent delays due to loading during game play. Finally each of the sound effects are tied to their own specific event whether that's a key up or key down movement from the player or collision with the obstacle or a reward. Thus in each of the different event listeners we have, if a sound effect is needed, we simply clone the Audio object of the corresponding sound and call play on it. We decided to clone the Audio objects instead of playing the same ones so that we could have overlapping sound effects when multiple different events are happening at once.

## Results

We largely measured success through our own experience with playing the game during its development. For instance, we iteratively adjusted the game speed whenever we felt that the game was unfeasible to make significant progress or that the game was too easy. Another measure of success is the visual aesthetics of the game; we paid close attention to the positioning, quantity, and appearance of background objects in order to strike a balance between constructing a visually interesting background

and not detracting from the main objective of the game. In particular, the animation speed of the background sea creatures in the ocean required significant fine tuning based on players' experience and feedback.

To test how intuitive the instructions and controls of *Triathlon Mania* are, we asked several friends to try the game. All of our users instinctively understood the three running and biking controls (i.e., moving left, moving right, and jumping up). They also appeared to enjoy playing the game, as they asked to play multiple times. However, our users often struggled with using the diving control during the swimming phase, as they forgot that the jumping movement (controlled by the up arrow) is replaced by a diving movement (controlled by the down arrow) during this phase. These results suggest that providing clearer instructions about the game controls, or even perhaps displaying a reminder during the swimming phase about the diving control, would improve the user experience.

## Discussion

Our approach of dynamically changing the terrain, obstacles, and player movements during a singular play of *Triathlon Mania* is promising, as our users enjoyed experiencing three distinct terrains and adapting their game strategy based on their current terrain. We are excited about the overall positive response to this game, and we are pleased with the creative direction and technical design decisions we pursued. Throughout this project, we faced challenges for which we had to assess technical and visual tradeoffs. Some of these challenges are outlined below:

1. **Detecting collisions:** Our game speed is slower than the rate at which the `onAnimationFrameHandler()` is called. Therefore, in our initial implementation of collision detection with bounding boxes, the player's collision with a singular obstacle was registered as several collisions. However, it is essential that each obstacle collision is only detected once to ensure that the player's number of remaining lives is accurate (the player's number of remaining lives is decremented upon every obstacle collision).
  - a. Solution: We maintain a Set of obstacles with which the player has collided recently. Upon an obstacle collision, we check if this set contains the given obstacle. If so, this obstacle collision has already been detected, and thus the player's number of lives should not change. The obstacle is removed from this set once it has passed the camera and is reset to its initial position. This approach is also adopted to keep track of valid reward collisions.
2. **Confining objects to the bounds of its corresponding terrain:** Due to its design, the `TerrainController` changes its terrain phase enum value halfway through the last lap of the current phase. In other words, the `TerrainController`'s phase state is inaccurate for the last  $\frac{1}{2}$  section of the current phase. This misalignment resulted in the appearance of obstacles, rewards, and background objects in the wrong terrain for a short period at the end of every terrain phase.
  - a. Solution: We add another state to `TerrainController` called `characterPhase`, which always stores the current terrain the player is in. When moving back an object's z position, we also check if the new z position is inside the bounding box of the next path. Objects that overlap with the next path are made invisible.
3. **Incentivizing risk taking for players:** To raise the stakes of our game, we added special properties to some of the rewards such that the player must make quick decisions on whether to attempt to reach the reward and risk their safety or to play it safe.

- a. **Levitating acorns:** To incentivize the player to jump over deer at the risk of mistiming their jump and, unfortunately, landing on a deer, we add acorns above select deer at a y position corresponding to the maximum height of the player's jump. Therefore, to reach a levitating acorn and avoid hitting a deer, the player must properly time their jump. We created a new FlyingAcorn class to implement this special property. To ensure that a levitating acorn is always positioned above a deer, we added a new state variable to the Deer class called hasAcorn. If hasAcorn is set to true, then a FlyingAcorn is passed into the Deer, and this object is updated to have the same x and z coordinates as the Deer whenever the Deer's position changes.
- b. **Bobbing treasure chests:** To increase the difficulty of collecting a swimming reward, we adjust the y position of the treasure chests according to a sinusoidal function, creating the illusion of the treasure chests naturally floating in the water. Therefore, to collect a bobbing treasure chest, the player sometimes must dive—while taking caution to avoid the swimming sharks.

Throughout this project, we gained valuable experience with game development, technical design, visual debugging, and Three.js. We enjoyed exploring the power of Three.js in this project, building on our Three.js knowledge from the course assignments. In addition, we learned how to animate 3D objects, both by using built-in animations via the Three.js AnimationMixer and by manually creating sinusoidal movements.

## Ethical Considerations

One ethical concern about *Triathlon Mania* is that the player avatar is not customizable. This fixed character is a white female, and thus everyone who plays the game must play as this character even if they would rather play as a character that better reflects their own identity with respect to skin tone, gender, or other aspects of their identity. This limitation clashes with the ACM Code of Ethics Principle 1.4: Be fair and take action not to discriminate, and Principle 3.2: Articulate, encourage acceptance of, and evaluate fulfillment of social responsibilities by members of the organization or group. Currently, players may feel a sense of bias by not being able to play a character that reflects their own racial identity. In this regard, *Triathlon Mania* fails to fulfill its relevant social responsibilities by not accommodating for different identities to feel represented in the player avatar.

Another objectionable aspect of *Triathlon Mania* is the risk of player obsession, as the premise of endless runner games is to keep players playing the game, sometimes to the point of addiction. This aspect relates to two ACM Code of Ethics principles: Principle 1.2: Avoid Harm, and Principle 2.5 Give comprehensive and thorough evaluations of computer systems and their impacts, including analysis of possible risk. If someone were to get addicted to our game, they could potentially suffer physical and/or mental injury, such as carpal tunnel syndrome in their fingers from the controls. Further, playing video games for over ten hours daily can also lead to a reduction in prefrontal cortex development in an individual's brain. Playing *Triathlon Mania* for prolonged periods could also damage the player's device, as it may be computationally expensive to keep the game running for long periods.

Follow-up work to improve these two aspects include adding customizable or more diverse characters to the game so that all players can find a character that they identify with. Customizable aspects could include the skin tone, hair color and length, and attire of the avatar. To address the risk of player addiction, *Triathlon Mania* could include a warning before the start of the game informing players

on the risks associated with getting addicted to playing the video game. In addition, the game could set a screen time limit, preventing users from playing the game for excessively long times.

## Conclusion

We effectively attained our goal of adding a fresh twist to traditional endless runner games by implementing dynamically changing terrain and character movements, as well as incentivizing risk taking. In addition to completing our MVP, we completed multiple stretch goals, including adding sound effects, implementing a dynamically changing background, and animating certain background objects. Future work for this project includes both user experience improvements and technical optimizations. Additional game features include adding customizable player avatars, keeping track of the player's score (which would correlate with the amount of time elapsed), adding new player "boosts" (such as an invincibility state or reduced obstacles state), and adding changing seasons or weather conditions in the background. Moreover, this game could be expanded into a multiplayer game to better simulate a true triathlon. Technical improvements to this game include further optimizing collision detection (i.e., discarding certain object pairs before checking bounding boxes), model loading (i.e., implement batch loading), and path movement. In addition, we can try out functions other than the sinusoidal function to more accurately simulate animal movements such as the dolphin jump. We can also work on further refining the positioning of objects in the scene.

## Contributions

Lauren created the models for the characters (runner, swimmer, biker) and animated their motions. She also implemented the character controls, specifically the jumping, diving, and switching lanes motions when an arrow key is pressed. She also implemented the character switching with new terrain functionality. She also added the lives tracker that dynamically updates upon every obstacle collision or reward collection and resets after the game restarts. She also created the home page and the game over page. She also worked on the interpolating sky colors. She animated and added cheering spectators in the running phase. She had a lot of fun studying different motions like jumping and swimming and undergoing trial and error with their coordinates.

Palzom found and implemented all of the sound effects for each of the different terrains and their respective obstacles and character movements. She also implemented the reward for the mountain path. She also added all of the background greenery to the mountain terrain, which included finding all of the 3D models, experimenting with different locations and offsets for aesthetics, as well as customizing different features of the models to better suit the terrain.

Note: Palzom got the flu in the beginning of the reading period. She was able to contribute to the brainstorming process in the beginning, but wasn't able to start coding until she had recovered and thus wasn't able to contribute a lot of implementation details to the final project itself.

Ruoming implemented the moving, infinite terrain and TerrainController, and she made the transitions between the terrains. She also created the different graphic effects for the three terrains (grass, ocean, mountains). She implemented player-obstacle, player-reward, and obstacle-reward collision detection. She implemented the deer, shark, and bird obstacles, including determining locations and offsets for the animals and animating the sharks and birds. She also implemented the rewards for the running and

swimming paths. She added background marine life to the ocean and animated these creatures in the swimming phase.

## References

- <https://medium.com/@joshmarinacci/low-poly-style-terrain-generation-8a017ab02e7b>
- <https://discoverthreejs.com/book/first-steps/animation-system/>
- <https://discourse.threejs.org/t/low-poly-water-surface-effect-for-compatibility-with-r155/54755>
- <https://discoverthreejs.com/tips-and-tricks/>
- <https://threejs.org/docs/>
- <https://medium.com/@joshmarinacci/low-poly-style-terrain-generation-8a017ab02e7b>
- <https://github.com/wanfungchui/Boxy-Run>
- <https://github.com/mrdoob/three.js/tree/master/examples/models/gltf>
- [https://threejs.org/examples/#misc\\_controls\\_pointerlock](https://threejs.org/examples/#misc_controls_pointerlock)
- <https://stackoverflow.com/questions/72527819/how-can-i-modify-a-material-of-a-gltf-model-in-three-js>
- <https://stackoverflow.com/questions/25165013/three-js-overlapping-layers-flickering>
- <https://29a.ch/simplex-noise/docs/modules.html>
- <https://poly.pizza/m/0tJzk22c46S>
- <https://poly.pizza/m/cQ-FnrdHHVw>
- <https://poly.pizza/m/f7SU-dO1FG7>
- <https://poly.pizza/bundle/Animated-Fish-Bundle-ZkGbjS8m8g>
- <https://poly.pizza/m/19VoUuA2pcN>
- <https://pixabay.com/sound-effects/sfx-mario-brothers-like-coin-grab-sound-effect-236423/>
- <https://pixabay.com/sound-effects/shark-eerie-scream-terrifying-growling-and-screaming-17-433581/>
- <https://pixabay.com/sound-effects/cartoon-trombone-sound-effect-241387/>
- <https://pixabay.com/sound-effects/jumps-65494/>
- <https://pixabay.com/sound-effects/bar-increase-cartoon-funny-jump-384919/>
- <https://pixabay.com/sound-effects/sub-dive-75777/>
- <https://pixabay.com/sound-effects/026499-crows-bird-squawking-fight-74501/>
- <https://pixabay.com/sound-effects/metal-hit-10-193281/>
- <https://poly.pizza/m/7I1IhiE7O8s>
- <https://poly.pizza/m/6Yjt8nIwLsD>
- <https://poly.pizza/m/RtLRqYjfMs>
- <https://poly.pizza/m/6pwjq7hSrHr>
- <https://poly.pizza/search/birds%20nest>
- <https://threejs.org/docs/?q=audio#Audio>
- <https://www.acm.org/code-of-ethics>

## Appendix A

