

Anha Khan  
 Dr. Adam Finkelstein  
 COS 426: Computer Graphics  
 13 December 2025

## Final Project Writeup – Kitty Courier

### I. Abstract

Kitty Courier is an endless runner style game, based in JavaScript under the Three.JS framework, in which the player controls a cat, named Bridget, who delivers parcels across a procedurally generated town. Players navigate sidewalks filled with various objects (e.g., foliage, like trees, and street decor, like vending machines, etcetera) and roads filled with speeding cars to pick up and deliver packages. This project uses the scaffolding code made available by Dr. Adam Finkelstein. Core contributions include a chunk-based terrain generation system, a character controller, character animation slicing, collision detection, a robust game loop, audio management, and user interface elements.

### II. Related Work

Kitty Courier combines elements of several games across varying genres. The project is inspired by endless runner games that I played growing up, like Subway Surfers and Temple Run. The game is also motivated by adventure games involving a cat protagonist navigating cities and obstacles, such as Stray and Little Kitty Big City. Combining the components of a cat avatar, a town with obstacles, and an endless runner, I present Kitty Courier!

### III. Approach and Methodology

The `SeedScene` class serves as the root of this game, as it initializes all the necessary assets: the cat model, the lights, the audio manager, a city asset manager, a terrain generator, and a game manager responsible for most elements contributing to the infinite gameplay loop (i.e., spawning parcel and delivery destinations, vehicle management and crash handling, score tracking, and game-over conditions). The class also manages the game's audio using Three.JS' `Audio`, `AudioListener` and `AudioLoader` libraries. All audio files used for this game have been downloaded from Pixabay; links to original sources are provided in the bibliography, with specific attributions listed in the `readme` of my Git repository.

The procedural generation of the game environment is made possible by the `City` and `Generator` class. The `City` class loads the .glb file containing all city assets (e.g., buildings, vehicles, and street decor, etcetera). Note, all 3D models used for this project have been downloaded from Sketchfab as .glb files; links to original sources are provided in the bibliography, with specific attributions listed in the `readme` of my Git repository. The city assets .glb file organizes everything as a scene graph, so the class traverses it using Three.JS' GLTFLoader, arranging each object into a category (e.g., a model of a coffee shop goes into the `building` category). The `City` class also consists of getter methods to retrieve a random object from each relevant category.

The `Generator` class then utilizes these categorized assets to create strips of terrain along the z-axis as the cat moves forward. In general, the world is modeled as a series of 44.7 by 44.7 unit chunks aligned along the z-axis. For a given chunk index, the class computes the chunk's world coordinates, populating it with a symmetric layout: a road running down the middle, grass tiles extending from the road towards the edges of the map along the x-axis, a random building placed on each sidewalk, randomized road decor to act as obstacles for when the cat is navigating towards a point of interest (POI) (i.e., to pick up or deliver a package), and fences running through the edges of the map to form a soft world boundary.

The `Cat` class implements the player's avatar, Bridget the kitty. The class loads the .glb file containing the model and its associated animations: idle, walk, and run. The class manages the cat's animations using Three.JS' `AnimationMixer`. The source file stores all motions in a single animation clip, so the class slices the clip into explicit time ranges, updating the animation mixer's time accordingly when switching states; animations are also infinitely looped. The `Cat` class implements movement by listening for keyboard events. 'W' and 'S' control forward and backwards movement, respectively, along the direction the cat is currently facing. 'A' and 'D' rotate the cat in place. And, holding 'Shift' while moving forward makes the cat run (i.e., move at a faster speed). Lastly, the `Cat` class implements collision detection with objects along the sidewalks using axis-aligned bounding boxes.

The `Game` class orchestrates all gameplay loop mechanics. The class manages package and delivery destination spawns; these POIs randomly spawn 60-90 units ahead of the cat. Parcels will spawn on either sidewalk at random, and the delivery destination will always spawn on the opposite sidewalk, forcing the player to cross the street every delivery cycle. The player automatically interacts with a POI (i.e., picks up or delivers their current package) as long as they are within 8 units of it. There is a red arrow that follows the cat, pointing the player in the direction of a POI. The `Arrow`, `Parcel`, and `Pointer` classes load the directional arrow, package, and delivery location pointer, respectively. The `Game` class also spawns vehicles at randomized intervals (every 1–4 seconds) at the front edge of the map along the z-axis, where they move towards the cat; there can be a maximum of 15 vehicles present in the world at a time to not overwhelm the player attempting to cross the road. Vehicle collisions with the cat are also managed using axis-aligned bounding boxes. If a car collides with the cat, the cat turns sideways, and the game ends.

Lastly, there are minimal graphical user interfaces within the game. Injecting HTML into `App.js` and using a CSS sheet, I made a start screen, an instruction screen, a score counter, an audio toggle button, and a game over screen. The icons used for the audio toggle button come from Iconify; links to the original sources are provided in the bibliography, with specific attributions listed in the `readme` of my Git repository.

#### **IV. Results, Limitations, and Future Work**

The resulting game implements a full game loop of picking up packages, crossing the street, and delivering packages, all wrapped in an environment that extends indefinitely as the player progresses. The cat's movement is quite smooth and visually intuitive due to its animations, and the proximity-based parcel pickup/delivery mechanism keeps interactions simple. Also, the procedural generation of the environment ensures each chunk varies in appearance slightly with random retrievals of buildings and street decor.

However, there exists some limitations. The use of simple hit boxes for collisions results in asymmetrical objects having blocking volumes larger than intended, causing the cat to collide with 'empty' space. For instance, a tree blocks a lot of space due to its crown despite the cat only being able to collide with its trunk. Moreover, game difficulty is static; there is no adaptive scaling as the player's score increases. Lastly, object movement is currently tied to frame rate instead of a fixed time step, so the game feels faster on displays with a higher refresh rate.

Future work can address these limitations. For example, we can integrate a physics engine to handle the complex collisions associated with the use of many asymmetrical objects in this game. We can also implement a strict, delta time-based update mechanism so that movement timing does not hinge on frame rate. Additionally, we can adaptively scale the game difficulty in

various ways as players increase their score, as we could increase the vehicle spawn rate, have vehicles begin spawning in both lanes, and/or have packages spawn in the road, etcetera.

## V. Ethical Concerns

One ethical concern that arises in the design of Kitty Courier is accessibility for players with color-vision deficiencies (e.g., red-green and blue-yellow color blindness), corresponding to ACM guidelines 1.2 avoid harm and 1.4 be fair and take action not to discriminate. The game mainly utilizes 3D assets with a low-contrast, pastel color palette. Thus, important gameplay elements, such as the package, delivery, and guidance markers, could be difficult to perceive for a player who is colorblind, thereby excluding them from having a fair experience with the game. To mitigate this concern, I can modify the aesthetics of Kitty Courier by incorporating non-color cues (e.g., distinct shapes) on essential pointers and/or offering selectable visual themes curated for common color-vision deficiencies, etcetera.

Another ethical concern that arises in the gameplay of Kitty Courier relates to the depiction of fictional animal harm, corresponding to ACM guidelines 1.1 contribute to society and to human well-being, acknowledging that all people are stakeholders in computing and 1.2 avoid harm. Currently, the game only ends when the cat experiences a collision with a car. Upon collision, a sound of a cat meowing plays, the cat flips onto its side, and the game ends with a message stating, “Bridget got hit by a car... :( But do not worry! She's a super cat who got up, unharmed, and went home safely.” Even though all models are highly cartoon-esque, and there is reassurance that Bridget is okay, some players could find this representation of a household pet colliding with a car upsetting. To mitigate emotional harm to players, I can adjust the fail-state of the game to be more ‘neutral.’ For example, on collision, I could incorporate an animation of the cat getting startled and retreating instead of being hit. I could also include a content warning on the instruction screen informing players of vehicle hazards.

---

*This report represents my own work in accordance with University regulations.*

*/s/ Anha Khan*

## Bibliography

- <https://annapurnainteractive.com/en/games-stray>
- <https://icon-sets.iconify.design/?query=volume&tag=Contains+Animations>
- <https://imangistudios.com/thegames/temple-run/>
- <https://www.littlekittibigcity.com/>
- <https://pixabay.com/vectors/triangle-polygon-background-2790346/>
- <https://pixabay.com/sound-effects/button-394464/>
- <https://pixabay.com/sound-effects/cat-meow-sound-383823/>
- <https://pixabay.com/sound-effects/clear-combo-7-394494/>
- <https://pixabay.com/sound-effects/click-1-384917/>
- <https://pixabay.com/sound-effects/double-car-horn-352443/>
- <https://pixabay.com/sound-effects/game-music-loop-6-144641/>
- <https://pixabay.com/sound-effects/walk-on-grass-1-291984/>
- <https://sketchfab.com/3d-models/car-arrow-c9f036364f274e8288249759c4be4516>
- <https://sketchfab.com/3d-models/low-poly-city-assets-3bb6c7a0db9d4acc8d8302932c0c2688>
- <https://sketchfab.com/3d-models/map-pointer-3d-icon-a30e2619537a425d90618ae5901c2989>
- <https://sketchfab.com/3d-models/parcel-d57214851ea44023b17639ccf4993cee>
- <https://sketchfab.com/3d-models/stripe-the-cat-rigged-and-animated-2e3030b71a6d4b219fdc7304f8e58013>
- <https://subwaysurfers.com/>
- <https://threejs.org/docs/>
- <https://threejs.org/docs/#AnimationMixer>
- <https://threejs.org/docs/#Audio>
- <https://threejs.org/docs/#AudioListener>
- <https://threejs.org/docs/#AudioLoader>
- <https://threejs.org/docs/#Box3>
- <https://threejs.org/docs/#Color>
- <https://threejs.org/docs/#GLTFLoader>
- <https://threejs.org/docs/#Group>
- <https://threejs.org/docs/#PointLight>
- <https://threejs.org/docs/#Scene>
- <https://threejs.org/docs/#TextureLoader>
- <https://threejs.org/docs/#Vector3>