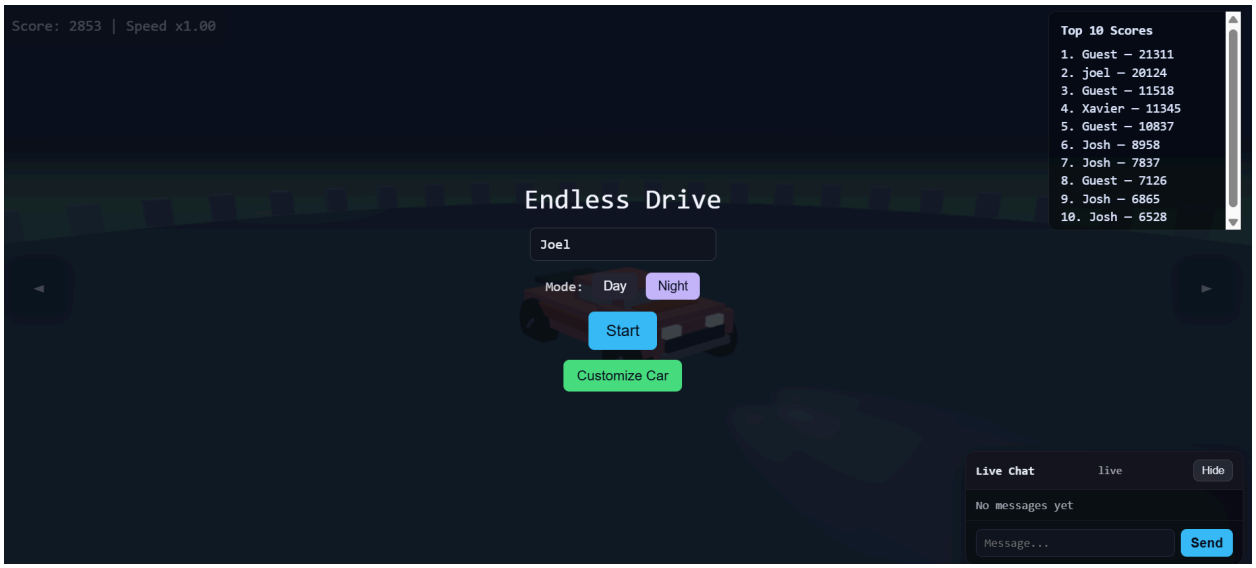
Joel Bustamante
jsb21j

Computer Graphics Final Project Report

Project: Endless Drive (minigame where goal is to drive as far as possible)

1. Project Overview

This project delivers an endless driving experience built with Three.js. The core loop procedurally spawns road segments, obstacles, and cross-traffic while the player auto-accelerates, steers to avoid collisions, and accumulates a speed-weighted score. A start menu provides car customization (color), day/night mode, orbit camera preview, leaderboard, and live chat. Mobile players get tap steering via full-screen touch zones.



2. Objectives

Create an endless, replayable driving scene with smooth controls and escalating difficulty.

Showcase graphics concepts: materials, lighting, fog, emissive elements, and simple model detail (cars, roads, foliage).

Implement procedural generation for roads, obstacles, and intersections.

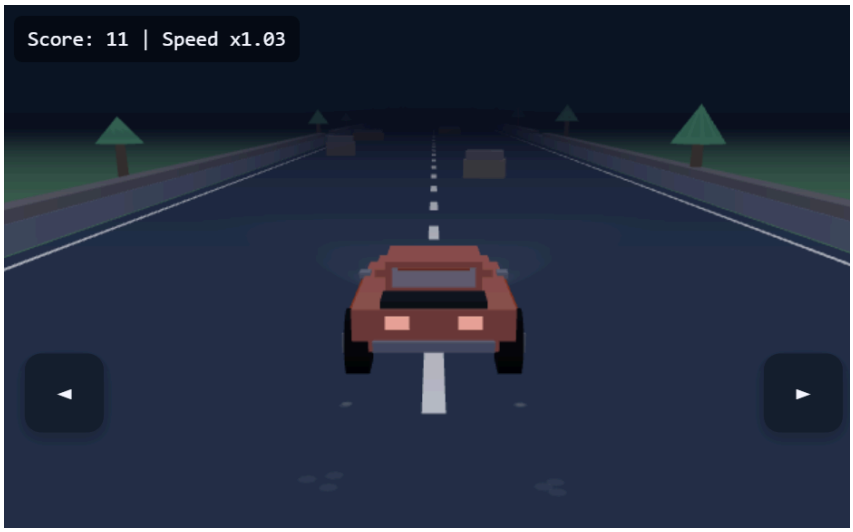
Integrate online features: leaderboard (Supabase) and live chat.

Support desktop and mobile inputs; keep performance acceptable on mid-range hardware.

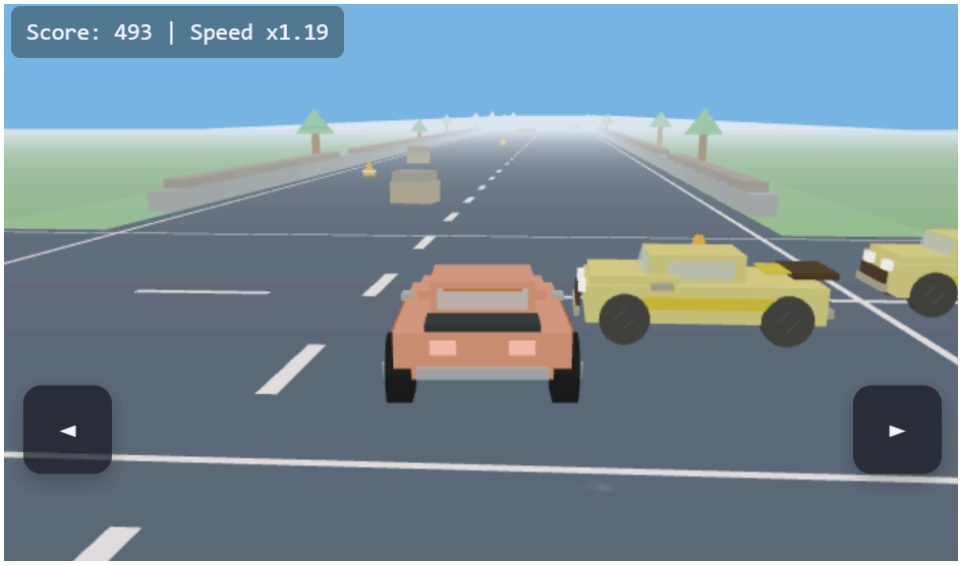
Joel Bustamante
jsb21j

3. Key Features

Procedural environment: Road segments stream in/out, with barriers, lane markings, grass shoulders, trees, and occasional intersections. Obstacles (cones/boxes) spawn with configurable density (current chance: 0.32).



Cross-traffic: Intersections spawn 1–3 AI cars with randomized timing/distance so they cross near the player with varied arrival times.

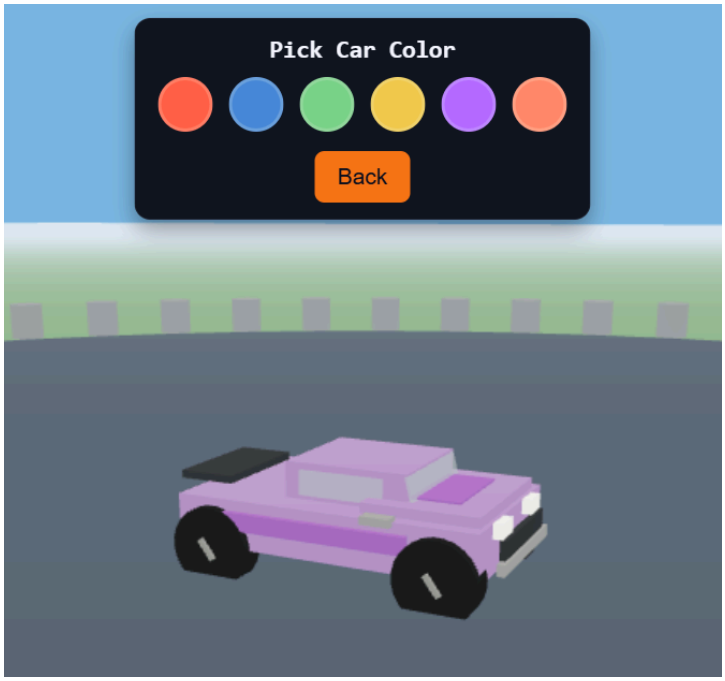


Vehicle systems: Auto-drive with ramping speed; steering responsiveness that scales with speed; collision detection against obstacles, barriers, and traffic; headlights (day/night); simple livery and tinted glass for both player and AI cars.

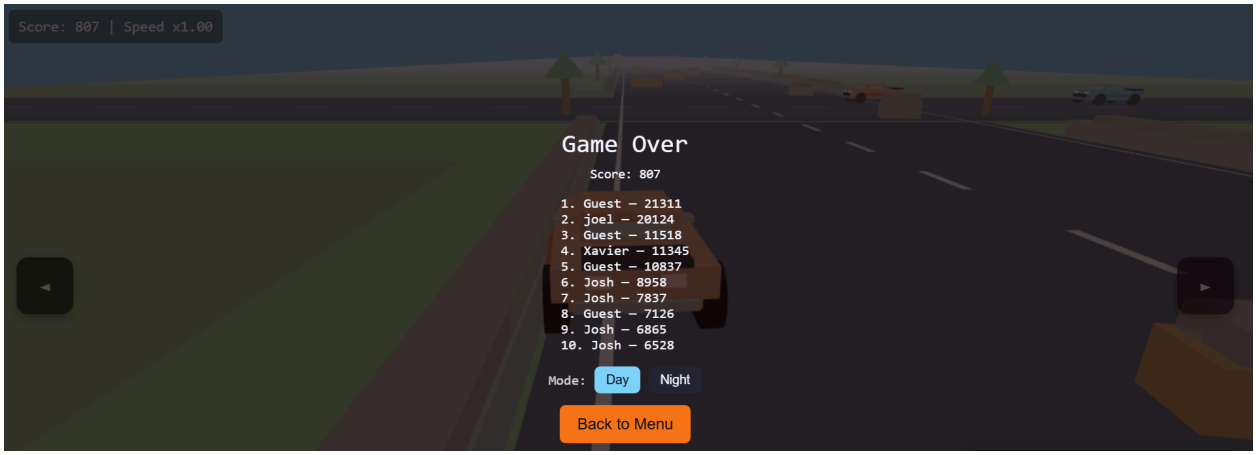
Scoring: Score accumulates per frame based on speed and distance, with a ramp-in period to slow early gain; HUD shows integer score and speed multiplier.

Joel Bustamante
jsb21j

UI/UX: Start menu with orbit camera around the car, color picker, day/night toggle, name input, and Start button;



Game Over overlay with score and leaderboard; mobile-friendly touch arrows plus tap zones.



Online services (Supabase): Leaderboard (top 10 scores) via REST; live chat via Realtime and a messages table with permissive RLS; scores saved on game over.

Joel Bustamante
jsb21j

4. Technical Stack & Architecture

Rendering: Three.js (0.116.0). Materials use MeshStandardMaterial for body/glass/wheels; emissive lights for head/tail lamps; fog per mode (Exp2).
Bundling: Webpack 4 with Babel (preset-env and optional chaining plugins); regenerator-runtime for async/await; pixel ratio capped at 1; shadows disabled by default for performance.

Procedural pipeline: The scene maintains a sliding window of segments (ahead/behind limits), spawning roads, obstacles, trees, barriers, and intersections; cleanup removes far segments and unregisters their objects.
Input: Keyboard (WD/Arrows), touch arrows, and full-screen left/right tap zones; menu orbit camera uses pointer drag.

State management: Scene-level state for game mode, camera mode, scoring, player input, car customization, overlays, and Supabase client/chat channel.

Networking: Supabase anon key/URL embedded (replace or move to env for production). Tables: scores (name, score), messages (id, user, body, created_at). Policies: allow anon select/insert (RLS enabled).

5. Implementation Details

Road & obstacles: spawnSegment builds road mesh, lane/side lines, shoulders, grass, trees, and barriers. Obstacle chance is 0.32 with lane jitter and separation. Trees spaced at 45 units to balance density/performance.

Intersections & traffic: Intersections have crossing roads and lane lines. spawnTrafficCars randomizes count (1–3), timing jitter, start distance, and speed (base

5.2–6.5 with small player-speed boost). Traffic activation triggers when the player is within a lead distance scaled by speed.

Vehicle modeling: Player/AI cars share body-colored cabins, roof panels, tinted front/rear/side glass, livery stripes, bumpers, grille, mirrors, and multi-part wheels. Headlights are spotlights toggled by mode; meshes for head/tail lamps match the player's car on AI.

Scoring loop: Each frame adds speed * delta * (1 + speed/maxSpeed) * ramp, where ramp reaches 1 after ~5s. HUD/game-over show floored score.

UI overlays: Start overlay with orbit camera, color picker (swatches), mode toggle, name input, Start button. Customize hides start overlay so only car + picker remain. Game Over overlay shows score, mode buttons, Back to Menu. Leaderboard appears on start and game-over overlays.

6. Performance Considerations

Shadows off by default; emissive lights used for lamps. Moderate tree density and obstacle chance tuned for performance. Pixel ratio capped at 1 to avoid high-DPI cost. Segment window limited to a few ahead/behind to keep object count bounded.

Materials lean on smooth/satin finishes without heavy postprocessing; no bloom/SSAO to keep frame times lower.

Chat/leaderboard network calls are lightweight; failures are ignored so gameplay isn't blocked.

7. Testing & Validation

Manual runs via npm start in Chrome/Edge desktop and mobile emulator:
Collision checks (obstacles, barriers, traffic) end the run and show Game Over.

Score increments faster at higher speeds; HUD shows integers; Game Over score matches HUD.

Traffic crosses with visible timing variance; no excessive bursts after limit reduction.
Start menu orbit and Customize picker: car color updates immediately; overlays hide/show correctly.

Touch steering works via tap zones and on-screen arrows.
Known limitations: Depends on Supabase availability; anon key in code (not ideal for production); no automated tests.

8. How to Run/Deploy
Install: npm install
Dev: npm start (webpack dev server at http://localhost:8080)
Build: npm run build
Deploy (GitHub Pages): npm run deploy (requires repo and gh-pages setup)
Supabase: ensure scores and messages tables exist with RLS enabled and permissive policies for anon select/insert; Realtime enabled for messages.

9. Challenges & Future Work

Balancing traffic timing so cars cross near the player without bursting too many at once.

Keeping performance smooth with procedural content; shadows remain off by default.

Future ideas: authenticated leaderboard, difficulty settings, audio (engine/collision), richer vehicle variety, tighter mobile UI, and moving keys/env out of code for security.