TECHNICAL REPORT

Interactive Graphics Course

Final Project

Master Degree in Artificial Intelligence and Robotics

Cyber-Tank Arena: Nomads

An Interactive 3D Browser Game Built with Three.js

2025

New content is underlined in red to show additions vs the original report.

1. General Overview

Cyber-Tank Arena: Nomads is an interactive 3D browser game developed using the Three.js library and JavaScript. The project presents a futuristic combat arena in which the player controls a cyber-tank set against a desert and steppe-inspired environment. The player's objective is to survive increasingly difficult enemy waves: destroying hostile drones, avoiding obstacles, and accumulating score and experience points.

The "Nomads" theme is reflected in the arena design, procedural ornaments, and the visual concept of obstacles inspired by traditional nomadic yurt structures. Key project highlights:

- Fully procedural textures generated in JavaScript, with no dependency on external image assets
- A complex hierarchical tank model featuring independent animations for each component
- A multi-layered lighting system with interactive headlights controllable by the player
- A wave-based enemy system with progressively increasing difficulty
- A modular code architecture in which each file handles a single, clearly defined aspect of the game
- A runtime configuration panel allowing players to customise colours, camera mode, and difficulty level

The game satisfies all core requirements of the Interactive Graphics course: hierarchical modelling, lighting, textures, user interaction, and JavaScript-based animations are all implemented directly in the codebase.

2. File Structure and Modules

The project is divided into independent modules, each with a clear area of responsibility. This separation makes the codebase easy to understand, extend, and maintain.

2.1 index.html

The main HTML entry file that creates the canvas element for 3D rendering and defines all interface elements. It contains HUD blocks for score, wave number, enemy count, XP, health bar, crosshair, headlight indicator, controls hint, and the game-over screen. It also loads Three.js, Tween.js, and the main JavaScript module.

2.2 style.css

Responsible for the visual appearance of the user interface: full-screen canvas, HUD positioning, orange cyber-style UI panels, health bar, crosshair, wave announcement, headlight indicator, control hints, and the game-over screen. The CSS gives the project a consistent cyberpunk and arcade aesthetic.

2.3 scene.js

Creates the main Three.js scene, renderer, and camera. Enables shadow mapping, sets the device pixel ratio, configures the background colour and scene fog, and handles window resize events. The camera uses a perspective projection and follows the player tank during gameplay.

2.4 textures.js

Generates all procedural textures through the HTML Canvas API without any external image files. For the tank it produces a base colour map (dark metal with scratches), a normal map (rivets and armour plate details), a roughness map (surface variation between matte and reflective zones), and an emissive map (glowing orange cyber-lines). A separate desert-like texture is created for the arena floor.

2.5 lights.js

Defines the complete lighting system: AmbientLight for baseline scene illumination, DirectionalLight acting as a sun or moon with shadow casting, a SpotLight functioning as tank headlights that follow the turret direction, and a PointLight used as a muzzle flash effect when the player fires. Headlights are toggled interactively with the L key.

2.6 tank.js

Implements the main hierarchical model of the project — the player tank. This module is discussed in detail in Section 3.

2.7 arena.js

Constructs the game environment: a large ground plane, glowing arena border walls, a star field, and static obstacles. It exports an obstacle list used for collision detection throughout the game. The obstacles are visually designed as futuristic blocks reminiscent of nomadic yurt structures.

2.8 input.js

Handles all user input: tank movement via WASD or arrow keys, turret aiming via mouse position, shooting via the left mouse button, and headlight toggling via the L key. The mouse cursor position is converted to a 3D point on the ground plane using raycasting, enabling accurate turret rotation.

2.9 bullet.js

Defines two bullet classes: PlayerBullet (fast, short lifetime) and EnemyBullet (slower, longer active). Both are represented as small glowing spheres with update() and destroy() methods. Bullets are removed upon hitting an obstacle, reaching their target, leaving the arena boundaries, or exceeding their lifetime.

2.10 enemy.js

Defines the enemy drone class. Each drone has its own 3D group containing a body, a glowing core, arms, and rotors. Drones spawn at random edges of the arena, move toward the player, rotate to face the tank, fire projectiles, and deal contact damage when close. Enemy speed scales with the current wave number to increase difficulty over time.

2.11 hud.js

Updates the user interface in real time: score, current wave, remaining enemy count, XP, health bar, wave announcement text, and the game-over screen. The health bar changes colour dynamically based on the player's remaining HP.

2.12 main.js

The central file connecting all modules and controlling the game loop. It manages game state, player HP, score, XP, wave number, enemy and bullet arrays, tank movement, shooting logic, collision detection, enemy and bullet updates, wave spawning, game-over handling, restart logic, and the rendering loop.

3. Hierarchical Tank Model

The primary hierarchical model of the project is the player tank, implemented in tank.js. The structure is designed so that each component can animate independently while inheriting transformations from its parent node. The object hierarchy is as follows:

- **tankGroup** — the root object of the entire tank. Controls the global position and rotation in the scene.
- **chassis** — attached to tankGroup. Rotates gradually to face the current movement direction.
- **wheels / treads** — child objects of the chassis. They move with the body and rotate when the tank is in motion.
- **turretGroup** — attached to the chassis but rotates independently to follow the mouse cursor for aiming.
- **barrelGroup** — child of the turret. Performs a recoil animation each time the player fires, implemented with Tween.js.
- **muzzleTip** — the spawn point for player bullets and the anchor for the muzzle flash light.

This hierarchy is significant because transformations propagate down the chain. When the root group moves, the entire tank moves with it. When the chassis rotates to face a new direction, the turret and barrel follow. At the same time, the turret can rotate independently for aiming and the barrel can animate its recoil without affecting the turret's orientation. The hierarchy eliminates the need to manually recalculate world-space coordinates for each part.

The wheels rotate in response to tank velocity, creating the visual impression of moving treads. The turret smoothly interpolates toward the mouse position each frame using the raycaster result. The barrel executes a backward-then-forward recoil tween on every shot, adding tactile feedback through animation.

The chassis also features a suspension wobble system: slight roll when turning and tilt when accelerating/decelerating, with spring-like interpolation for smooth physical feedback.

The tank uses PBR (Physically Based Rendering) materials with metalness 0.7, roughness 0.6, and emissive intensity 0.4, producing a convincing metallic surface that responds naturally to all four light sources.

4. Lighting System

The project uses four types of Three.js light sources, working together to produce a rich and atmospheric visual scene.

4.1 AmbientLight

Provides uniform base illumination across the entire scene. Without this source, surfaces facing away from the directional light would be completely black. It has no direction and casts no shadows; it simply ensures a minimum level of visibility for all objects in the scene.

The ambient light colour dynamically shifts toward blue/purple hues in higher waves, creating a progressively darker and more intense atmosphere as difficulty increases

4.2 DirectionalLight

Simulates a distant sun or moon above the steppe arena. This is the primary shadow-casting light: all objects project shadows in a single consistent direction, giving the scene depth and volume. It is positioned high above the arena and angled downward.

Uses PCFSoftShadowMap for soft blurred shadow edges, with a 4096×4096 shadow map resolution and an orthographic frustum covering ±45 units.

4.3 SpotLight — Tank Headlights

Conceptually attached to the tank turret, following the aiming direction of the barrel. It produces a conical beam of light in front of the tank. The player can toggle the headlights on and off with the L key, demonstrating real-time interactive control over a scene light source. A HUD indicator shows the current state of the headlights.

The headlight uses a 512×512 shadow map, penumbra of 0.4 (40% soft edge), and a visible transparent cone mesh (CylinderGeometry, opacity 0.06, DoubleSide) attached to the headlight mount so the beam volume is visible to the player.

4.4 PointLight — Muzzle Flash

Used as a muzzle flash effect when the player fires. On each shot, the light intensity jumps sharply to a high value and then fades back to zero. This transition is implemented as a Tween.js animation on the light's intensity property. The effect adds a visually convincing and atmospheric shot feedback to the gameplay.

In addition to the PointLight, a Sprite with additive blending and a procedural radial gradient texture (yellow centre to transparent edge) is positioned at the muzzle tip and faded out over 80 ms via Tween.js, creating a visible flash orb.

5. Textures and Materials

All textures in the project are generated procedurally using the JavaScript HTML Canvas API. No external image files are required, making the project entirely self-contained and demonstrating mastery of programmatic texture generation.

5.1 Base Colour Map

Creates the primary visual appearance of the tank hull: a dark metallic base with scratches, dents, and decorative lines reminiscent of nomadic ornaments. The surface looks worn and battle-hardened, consistent with the cyberpunk aesthetic of the project.

5.2 Normal Map

Simulates surface relief detail without adding geometry. It creates the illusion of rivets, armour plates, and fine surface irregularities. Three.js uses this map to calculate per-pixel lighting based on the encoded surface normals, significantly enhancing the perceived detail of the tank mesh.

5.3 Roughness Map

Controls how specularly reflective each area of the surface appears. Low-roughness regions simulate polished metal that catches light, while high-roughness regions imitate matte combat damage and worn paintwork. This per-pixel variation makes the material feel realistic and physically plausible.

5.4 Emissive Map

Adds glowing orange lines and symbols to the tank body. Because the emissive component is independent of external lighting, these details remain visible even in unlit conditions. They reinforce the cyberpunk visual identity of the tank and make it easily distinguishable from the environment.

5.5 Ground Texture

A procedural texture applied to the arena floor, designed to resemble a desert or steppe landscape. It gives the combat arena a natural-looking surface that complements the nomadic theme of the project.

6. User Interaction

The game supports multiple forms of user interaction with both the 3D scene and the interface layer.

6.1 Movement

The WASD keys or arrow keys control the tank’s movement across the arena. The tank responds with smooth acceleration and deceleration when keys are pressed and released, giving the vehicle a sense of physical weight. The chassis gradually rotates to align with the direction of travel.

6.2 Mouse Aiming

The mouse cursor position is converted to a 3D world-space point on the ground plane using Three.js Raycaster. The turret smoothly rotates toward this point each frame, providing intuitive and decoupled aiming: the tank body and the turret can face different directions simultaneously.

6.3 Shooting

Pressing the left mouse button fires a projectile in the direction the turret is facing. Each shot triggers the barrel recoil animation and the muzzle flash point light effect.

Shooting is rate-limited to once every 250 ms (4 shots per second) to prevent excessive fire.

6.4 Headlight Toggle

The L key toggles the SpotLight headlight on and off. The current state is reflected immediately in the HUD indicator. This feature demonstrates real-time interactive control over a scene light source and is a direct response to player input.

6.5 Interface Reactivity

The HUD is updated every frame to display the current score, wave number, remaining enemy count, XP total, and health bar. When the player is killed, the game-over screen appears with final statistics and a restart button that fully resets the game state.

6.6 Configuration Panel

Pressing the Tab key opens a configuration panel that pauses the game. The panel provides four colour pickers (tank body colour, bullet colour, ambient light colour, and HUD accent colour), a camera mode selector (Default / Chase / Orbit), and a difficulty selector (Easy / Normal / Hard). All changes are applied in real time. The panel uses the same cyberpunk aesthetic as the rest of the UI with orange borders and dark translucent backgrounds

6.7 Wireframe Toggle

Pressing the X key toggles wireframe rendering on every mesh in the scene by traversing the entire Three.js scene graph. This is useful for debugging and for inspecting the geometric structure of models during development

7. Animations

The project contains a wide variety of animations implemented in JavaScript. Smooth transitions are handled by Tween.js where precise interpolation is required.

7.1 Tank Animations

- Movement: smooth acceleration and deceleration applied each frame based on key input state.
- Chassis rotation: the hull gradually interpolates toward the current movement direction.
- Wheel rotation: wheels spin in proportion to the tank's speed, simulating moving treads.
- Turret rotation: the turret smoothly follows the mouse cursor position each frame.
- Barrel recoil: on firing, the barrel translates backward and returns to rest using a Tween.js sequence.
- Muzzle flash: PointLight intensity spikes and fades back to zero via a Tween.js animation.

Chassis suspension: the hull rolls slightly when turning and tilts when accelerating or decelerating, with spring-like interpolation for physically convincing weight transfer

7.2 Enemy Animations

- Pursuit movement: drones continuously move toward the player's current position.
- Target tracking: enemies rotate each frame to face the tank.
- Shooting: drones periodically fire projectiles toward the player on a timed interval.
- Core rotation: the glowing central core of each drone spins continuously.

Hover bob: each drone hovers above the ground with a vertical sine-wave oscillation.

Rotor spin: the four rotors at the arm tips spin at 8 rad/s, producing a convincing hovering drone aesthetic.

Arm wobble: the four arms oscillate with sine-wave offsets for a mechanical vibration effect.

Core pulse: the glowing core scales between 0.9 and 1.1 in a breathing rhythm.

Hit flash: when struck, the core emissive intensity spikes for 0.15 s to provide immediate visual feedback.

Death animation: on destruction, a PointLight flash (intensity 6→0 over 300 ms) accompanies a scale-shrink tween (1→0 with Quadratic.In easing), after which the drone is removed from the scene.

7.3 Bullet Animations

Player and enemy projectiles move through the scene on every frame tick, producing a dynamic combat atmosphere. Bullets are removed immediately upon hitting an obstacle, a target, an arena boundary, or expiring their lifetime timer.

Each bullet carries a trailing line rendered as a BufferGeometry with manually updated vertices per frame: player bullets use a 5-point trail buffer (orange, opacity 0.3) and enemy bullets use a 3-point trail buffer (red, opacity 0.25). An additive-blended glow Sprite with a procedural radial gradient texture is attached to each bullet for a visible halo effect

7.4 Camera Animation

The camera smoothly follows the tank by interpolating its position toward a target offset above and behind the tank each frame. This approach prevents abrupt camera cuts and creates a more dynamic third-person / top-down hybrid viewing experience.

Three camera modes are available: Mode 0 (Default) — top-down follow with smooth interpolation behind the tank; Mode 1 (Chase) — close behind the tank, following the last movement direction; Mode 2 (Orbit) — the camera circles continuously around the tank at a fixed radius of 10 units. The active mode is selectable from the configuration panel

7.5 Wave Transitions

When all enemies in a wave are destroyed, an announcement text fades in on the HUD. A short delay passes before the next wave spawns, giving the player a brief moment to prepare and adding visual rhythm to the game’s pacing.

7.6 Arena Animations

Obstacle hover bob: each of the 13 obstacles oscillates vertically with a sine wave (unique phase offset per obstacle).

Obstacle rotation: obstacles rotate slowly around their Y axis (0.05–0.15 rad/s).

Obstacle emissive pulse: the orange emissive glow intensity pulses over time.

Starfield twinkling: the 1 200-point star particle system has animated opacity (twinkling) and slow Y-axis rotation.

Fog: exponential fog (FogExp2, density 0.018) matching the dark blue background colour, creating seamless horizon blending

8. Visual Effects and Rendering

This section covers additional visual features not addressed in the original report.

8.1 Tone Mapping

The renderer uses Cineon tone mapping at exposure 1.2, producing filmic colour reproduction that enhances the cinematic feel of the scene. Combined with PCFSoftShadowMap shadow edges, the result is a visually polished output

8.2 Low HP Vignette

When the player's HP drops below 25%, a red vignette overlay (CSS border + inset box-shadow) appears around the screen edges, and the HP bar pulses with a keyframe animation. This provides immediate peripheral awareness of critical health without requiring the player to read the HP number

8.3 Invulnerability Visual Feedback

After taking damage, the player gains 0.5 seconds of invulnerability. During this period the chassis material emissive intensity flashes rapidly between high and low values, giving clear visual confirmation that the player cannot be harmed

8.4 Enemy Death Flash

When an enemy is destroyed, a PointLight (intensity 6, distance 10) spawns at the drone's position and fades to zero over 300 ms via Tween.js, creating a dramatic explosion-like flash before the drone model shrinks and is removed

9. Game Mechanics and Conclusions

9.1 Wave System

The game is structured around a wave-based progression system. The first wave spawns five enemies. After the player destroys all enemies in a wave, the wave counter increments and a new wave begins after a short delay. Each subsequent wave spawns more enemies and increases their movement speed, creating a steady escalation of difficulty over time.

The exact formula is 5 + wave × 2 enemies per wave (e.g., wave 1 = 5, wave 5 = 15, wave 10 = 25). Enemies spawn one at a time with a 400 ms stagger to avoid frame-rate spikes.

9.2 Health and Scoring

The player starts with 100 HP. Health decreases when enemy projectiles strike the tank or when a drone makes direct contact. Destroying enemies awards score points and XP. When HP

reaches zero the game ends and the final score is displayed. The player can restart at any time, which fully resets score, XP, HP, wave number, tank position, and all active bullets and enemies.

Score per kill: 100 + wave × 10 (e.g., wave 1 kill = 110 points, wave 10 kill = 200 points). XP per kill: 25. Enemy bullet damage: 8 HP. Contact damage: 15 HP per second. Invulnerability window: 0.5 s after each hit.

9.3 Enemy Progression

Enemies have three visual tiers that change with wave progression: Tier 0 (waves 1–3) — red body and core; Tier 1 (waves 4–6) — purple body and core; Tier 2 (waves 7+) — white/chrome body and white core. Each tier has distinct visual identity, making it immediately clear to the player that difficulty has increased. Enemy base HP scales with difficulty setting (ceil(2 × difficulty)), and speed scales with both wave number and difficulty. Shoot intervals are also shortened at higher difficulties, with a minimum of 0.5 seconds between shots

9.4 Difficulty System

The game offers three difficulty levels selectable from the configuration panel: Easy, Normal, and Hard. The difficulty multiplier affects enemy HP (higher = more hits to destroy), enemy speed (higher = faster pursuit), and shoot rate (higher = more frequent enemy fire). This allows players of different skill levels to enjoy the game and provides replay value

9.5 Collision Detection

Custom collision detection is implemented without an external physics engine.
Circle-to-rectangle intersection tests are used to prevent the tank from passing through obstacles and to stop bullets on impact. Bullet-to-target collisions are resolved using distance comparisons. This approach is intentionally simple and fast enough for the scale of the game.

The collision system uses squared-distance checks (no square root) for performance optimisation. Enemies also perform obstacle avoidance when tracking the player, steering around blocks instead of getting stuck.

9.6 Fulfilment of Course Requirements

The project fully satisfies the requirements of the Interactive Graphics course:

- Hierarchical modelling: a complex five-level tank hierarchy with independent animations at each level.
- Lighting: four light types in active use — AmbientLight, DirectionalLight, SpotLight, and PointLight.
- Textures: five texture maps applied — base colour, normal, roughness, emissive, and ground texture.
- User interaction: keyboard movement, mouse aiming, shooting, light toggling, HUD reactivity, and restart.
- Animations: more than ten distinct animation types implemented in JavaScript and Tween.js.

In conclusion, Cyber-Tank Arena: Nomads is a fully playable interactive 3D game that demonstrates all of the key concepts required by the course. The modular code structure clearly

separates concerns across files, while the gameplay experience goes well beyond a static 3D scene, offering wave-based combat, a progression system, health management, collision handling, and a complete restart flow. The procedural approach to texture generation and the depth of the hierarchical model further distinguish the project as a technically thorough implementation of the course material.