If you’re reading this, then this was a conversation I had with an LLM about what I wanted the game to be. It will give you insight into the overall project and its goals. Some changes have been made since the making of this document. The map size has been cut in half, instead of 131072 x 131072, the map is now 131072 x 73729 pixels.

🚀 Space Bastard – Core Game Architecture & Roadmap (v2 - Fresh Start)

This is the definitive development document for the new, from-scratch version of *Space Bastard*, a physics-driven tactical space combat sim. You are currently building a combat prototype that emphasizes **real scale**, **real physics**, and **true spatial awareness** — not abstract "strategy game" zoom grids.

We’re building toward a full FTL-like loop down the road, but for now: just the hard, beautiful truth of space combat.

🌌 CORE CONCEPT

The map is **131,072 x 131,072 pixels**, where each pixel = **0.25 meters**. This gives us a real-world map size of:

- **32,768 meters (32.768 km)** per edge from center
- **65.536 km x 65.536 km** total field
  This is *not* a “zoom illusion” — this is the actual world size in which ships, torpedoes, and bullets live.

All gameplay occurs in this space, rendered via **one unified 2D layer**, with on-screen UI only. The player zooms the view (via `Camera2D.zoom`) to navigate it, not the other way around.

🧱 ARCHITECTURE (NODE TREE + STRUCTURE)

We keep things clean and simple.

🔸 Scene: `WorldRoot.tscn` (Main Scene)

This is the root node of the entire game. It contains everything.

scss
CopyEdit

```
WorldRoot (Node2D)
├── GameCamera      (Camera2D)        ← Zoomable player view
├── GridOverlay     (Node2D)          ← Infinite adaptive grid
├── PlayerShip      (Area2D/Node2D)   ← Player vessel
├── EnemyShip       (Area2D/Node2D)   ← Target ship
├── TorpedoManager  (Node2D)          ← Spawns & tracks torpedoes
├── UI              (CanvasLayer)     ← Minimal UI on screen (buttons,
readouts)
└── Background       (ColorRect or Parallax) ← Empty space
```

📐 SCALING & WORLD SETTINGS

`WorldSettings.gd`

This autoloaded singleton stores core values:

gdscript
CopyEdit

```
var meters_per_pixel: float = 0.25
var map_size_pixels: int = 131072
var map_size_meters := map_size_pixels * meters_per_pixel
```

This sets the stage for **true physics** over **real scale**.

🔭 CAMERA ZOOM

The player can zoom smoothly in and out with the mouse wheel.

- **Zoom range**: From full map view (fit ~65km) to ultra close-up (~10cm resolution)
- Implemented in `GameCamera.gd`:
    - `zoom = Vector2.ONE * new_zoom` (min/max enforced)
    - Can eventually zoom to mouse cursor, but initially just center-based

---

⚙️ SYSTEMS (Core Gameplay)

✅ Physics-Based Ships

- `PlayerShip` and `EnemyShip` are fully physics-aware
- Movement is driven by force/impulse, not instant motion
- Ships will **never** just "go to target" — you fire RCS or burn fuel to move

✅ Torpedoes (Phase 1)

- Accelerate in straight lines with fixed thrust
- Have rotational inertia (can't instantly rotate)
- Use own sensor data to predict intercepts
- Calculations run in `_physics_process()`, with delta time scaled to real seconds

✅ Radar / Lidar (Phase 2)

- Torpedoes/ships don't "magically know" target location
- They scan using:
    - **Radar**: Slow update, wide angle, long range

    - **Lidar**: Fast update, narrow angle, short range
- They build a local model of enemy location with confidence values (color-coded icons)
- Old data fades over time

✅ Interception & Misses

- Torpedoes **can miss**. They may never reacquire a target.
- Every intercept is based on physics, not cheat-seeking
- Evasive maneuvers from ships matter

🔫 POINT DEFENSE SYSTEM (Phase 3)

Ships will automatically fire high-velocity bullets (PDC) at incoming torpedoes.

- Bullets are very fast, unguided
- Modeled as hitscan or fast-moving `Area2D`
- Use raycasts or bullet velocity logic to simulate near-instant kill windows
- Eventually include ammo tracking & blind spots

💥 RAILGUNS (Phase 4)

- Fires solid kinetic rounds at absurd speed (e.g., 10,000+ m/s)
- No homing, just unguided slugs
- Useful for close-range kills or precision long-range strikes
- Will require high lead accuracy from the player

🛠️ ROADMAP (Chronological Build Order)

🔹 Phase 1: Base Setup

- Setup `WorldRoot.tscn` with ships, camera, and grid
- Implement `WorldSettings.gd` to handle meters-per-pixel
- Setup zooming with `Camera2D.zoom` (centered only)
- Add ColorRect background to visualize map size
- Add basic movement controls for ships
- Add torpedo launch system

🔹 Phase 2: Real Physics & Intercepts

- Simulate acceleration-based torpedoes
- Use simplified constant-thrust navigation
- Detect proximity detonation

🔹 Phase 3: Sensors & Targeting

- Add radar/lidar systems
- Track scan confidence over time
- Display ghost/uncertain icons

🔹 Phase 4: Advanced Torpedoes

- Add roles: Interceptor, ECM, Escort, Fragmentation
- Add logic modules per torpedo type

- Animate fragmentation near target

🔹 Phase 5: PDC Systems

- Target incoming threats
- Add delay/accuracy per weapon
- Display tracers and intercepts visually

🔹 Phase 6: Railguns & Manual Firing

- Implement railgun with lead prediction
- Add UI element to show predicted collision
- Visualize shots and impacts

🔹 Phase 7: Visualization & QoL

- Add zoom-to-cursor
- Add picture-in-picture of tracked torpedoes
- Add slow-motion playback when impacts occur
- Grid dynamically adjusts size and step to zoom

🧪 FUTURE: What This Demo Will Teach Us

- How feasible it is to run high-object-count space combat on a large map in 2D
- Whether sensor-driven combat feels satisfying and unique
- If we can make realistic systems intuitive enough for players to enjoy

---

🎯 GOAL: A Launchable Demo

This prototype will not have:

- Story
- Win conditions
- Meta progression

But it *will* allow you to:

- Watch 10+ torpedoes race across a massive battlefield
- See if they land hits or miss entirely due to bad data
- Test the limits of Newtonian tactics on a real scale map