Project Goal

Build a **Minecraft-like voxel game in Java** that is: - Deterministic (fixed-rate ticks) - Scalable (millions of blocks, infinite world) - Data-oriented (IDs + arrays, not objects) - Cleanly layered (logic, rendering, saving separated) - Beginner-implementable, but not beginner-limited

This document is the **single source of truth** for architecture, ownership, and direction.

Core Design Principles (Non-Negotiable)

1. **Fixed timestep simulation** (ticks ≠ FPS)
2. **World owns all mutable game state**
3. **Blocks are data IDs, behavior is shared**
4. **Rendering is read-only**
5. **Saving is chunk-based and binary**
6. **Strict dependency direction (no cycles)**
7. **Packages enforce architecture**

Entry Point

App

Package: `com.game`

Role: - JVM entry point - Creates and wires systems - Starts game loop

Rules: - Contains no game logic - Nothing depends on it

Time & Execution

GameLoop

Package: `com.game`

Role: - Measures real time - Maintains accumulator - Executes fixed-rate ticks - Calls render with interpolation

Rules: - No world logic - No rendering logic

TickManager

Package: `com.game.tick`

Role: - Executes tickable systems in deterministic order

Contains: - List of `Tickable`

Rules: - Single-threaded - No rendering - No I/O

Tickable (interface)

Package: `com.game.tick`

Role: - Contract for fixed-rate updates

Implemented by: - World - EntityManager - Systems (later)

World Layer (Owns All State)

World

Package: `com.game.world`

Owns: - ChunkManager - EntityManager - BlockEntityManager

Role: - Coordinates world-level logic - Delegates ticking

Saved: - Seed - World rules

ChunkManager

Package: `com.game.world`

Owns: - Loaded chunks - Active chunk set

Role: - Load/unload chunks - Decide which chunks tick

Chunk

Package: `com.game.world`

Owns: - Block ID array - Light arrays - Scheduled block ticks

Role: - Raw data container - Executes scheduled updates

Rules: - No block behavior - No rendering

Saved: - Block IDs - Light data - Tick queues

Blocks (Behavior Only)

Block (abstract)

Package: `com.game.block`

Role: - Defines block behavior rules - Shared flyweight

Does NOT: - Store position - Store world data

BlockRegistry / Blocks

Package: `com.game.block`

Role: - Maps ID → Block - Holds static block definitions

Rules: - IDs are stable forever - Registry is never saved

Block Entities (Extra State)

BlockEntity (abstract)

Package: `com.game.blockentity`

Used for: - Chests - Furnaces - Machines

Owns: - Extra per-block state

Saved: - Position - Type - Custom data

Entities

Entity (abstract)

Package: `com.game.entity`

Owns: - Position - Velocity - State

Role: - Game object logic

EntityManager

Package: `com.game.entity`

Owns: - Active entities

Role: - Tick entities - Add/remove entities

Player

Package: `com.game.entity`

Role: - Controlled entity

Rendering (Read-Only)

RenderEngine

Package: `com.game.render`

Role: - Interpolates state - Draws world and entities

Rules: - Never mutates world - Never ticks

Saving & Loading

SaveManager

Package: `com.game.save`

Role: - Owns disk format - Saves/loads chunks and entities

Rules: - World never touches disk - Binary, chunk-based - Versioned

---

Utility Layer

util

Package: `com.game.util`

Contains: - Vector math - Bounding boxes - Helpers

Rules: - Depends on nothing - Used by everything

Package Dependency Direction (Must Hold)

```
util
 ↑
tick    block
  ↑       ↑
 entity   blockentity
     ↑        ↑
        world
           ↑
        render   save
             ↑
            App
```

If a dependency points the wrong way, the design is broken.

What Gets Saved

Data	Owner	Saved
Block IDs	Chunk	Yes
Block behavior	BlockRegistry	No
Entities	EntityManager	Yes
Block entities	BlockEntityManager	Yes
Render meshes	RenderEngine	No

Performance Rules

- Blocks = primitive IDs
- Chunks = flat arrays
- Tick only active chunks
- Schedule block updates
- Save only dirty chunks

Build & Tooling

- **Maven** project
- Single module initially
- Git from day one
- No Java serialization

Things Deferred Until Later

- Lighting engine
- Fluids
- Redstone-like logic
- Multiplayer
- Modding
- Shaders

Success Milestones

1. World loads
2. Chunks tick
3. Blocks persist after restart
4. Player moves
5. Save version survives changes

Mental Model Summary

- World = state
- Blocks = rules
- Ticks = time
- Rendering = view
- Saving = persistence
- Packages = contracts

This architecture is intentionally boring — because boring scales.