Columnar Game Logic

Empirical Validation of DataFrame Architectures for Real-Time Simulation

Xuanlin Zhu

January 2026 · Empirical Systems Research

Abstract

Entity Component Systems (ECS) have become the standard for high-performance game simulation, advocating Data-Oriented Design over traditional Object-Oriented Programming. However, many ECS implementations rely on custom, engine-specific memory management. This study proposes **Columnar Game Logic**—a complementary approach that implements ECS principles using mature Data Science tooling (NumPy, Polars, Apache Arrow). By treating game state as DataFrames rather than bespoke memory pools, we gain interoperability with the data science ecosystem while maintaining real-time performance. Benchmarks validate **500,000 entities at 60 FPS** using a Stratified Architecture that bridges the gap between data analysis assumptions (immutability) and game engine requirements (zero-allocation loops).

1. Motivation: The Simulation Gap

Three observations motivate this work:

Emergent Complexity. Simulation-heavy architectures produce hard-to-trace emergent behaviors. The famous *Dwarf Fortress* "drunken cats" bug—where cats died from alcohol poisoning after licking spilled beer from their fur—illustrates how complex rule interactions create debugging challenges. This work uses *data-centric tooling* (Arrow export, web visualizers) to make such emergent behaviors inspectable post-hoc, rather than requiring engine-specific debugging infrastructure.

Reinventing the Wheel. Game engines often implement custom columnar storage and SIMD-optimized loops from scratch. Meanwhile, the data science ecosystem offers mature, battle-tested implementations: NumPy for vectorized arithmetic, Polars for query optimization, and Apache Arrow for zero-copy interoperability across languages (Python, Rust, JavaScript). This project demonstrates that these tools can handle real-time simulation workloads.

OOP Performance Limits. Many simulation games—particularly 1990s-era tycoon and management games—suffer from performance degradation due to legacy Object-Oriented architectures. Our benchmarks show that switching from OOP to Columnar/ECS patterns yields up to **23.9**× speedup for arithmetic-heavy logic, validating the importance of Data-Oriented Design for large-scale simulation.

2. Empirical Findings

Six experiments establish the viability of DataFrame-based game logic:

Finding	Metric	Implication
Crossover point	1K–10K entities	DataFrame overhead dominates below 1K; vectorization wins at scale
NumPy in-place	**3.7**× vs. Polars chains	Use `out=` parameter for hot paths
Dirty tracking	**115.6**× upload reduction	Version-based skip avoids redundant GPU transfer
Stratified stability	0.89× creep factor	Frame times *improve* over time at 100K scale

Scaling behavior. At 100K entities, the DataFrame approach achieves **23.9**× over OOP for pure arithmetic (L0), **15.3**× for state machine logic (L5), and **4.8**× for branching conditionals (L1). The crossover point—where DataFrame overhead is amortized—occurs between 1,000 and 10,000 entities.

Logic creep analysis. A 500K-particle render pipeline showed frame time growth from 3.4ms to 13.7ms over 660 frames. Investigation confirmed this was *physics-correct* (boundary accumulation increasing collision checks), not architectural degradation. The Stratified Architecture showed stable or *improving* frame times (creep factor < 1.0) in controlled experiments.

3. Technical Architecture: Stratified Tables

To adapt immutable DataFrames for real-time (60Hz) loops, this project introduces a **Stratified Architecture** that separates update frequencies:

Path	Tooling	Frequency	Latency
Hot (physics)	NumPy `out=`	Every frame	0.35ms
Warm (game logic)	Polars batch	Periodic	1.25ms
Cold (persistence)	DuckDB/SQL	On-demand	Variable

The key insight: Polars always allocates new memory (even with `ref_count=1`), but the underlying NumPy arrays *are* shared. By extracting NumPy views for hot-path physics and using explicit version tracking (`col.version`) for dirty detection, we achieve the zero-allocation guarantees that game engines require while retaining DataFrame ergonomics for complex queries.

4. Future Work: PhD Vision

The use of Apache Arrow as the underlying memory format enables a migration path toward browser-native and GPU-accelerated simulation:

Phase 1 (Complete). Arrow IPC export for web visualization. Simulation frames are serialized to `.arrow` files consumable by JavaScript (Deck.gl, Observable) without Python runtime dependencies.

Phase 2 (In Progress). Rust/WASM hot paths. Porting NumPy-equivalent logic to Rust enables SIMD-optimized simulation in the browser via WebAssembly, with Arrow providing zero-copy data sharing between Rust and JavaScript.

Phase 3 (Proposed). R-tree spatial queries. Integrating spatial indexing (via DuckDB's `spatial` extension) would enable efficient neighbor lookups for agent-based simulations—currently the primary bottleneck for scaling beyond 500K entities.

Phase 4 (Proposed). WebGPU compute shaders. Moving physics entirely to WGSL compute shaders would eliminate CPU-GPU transfer, potentially enabling 5M+ entities by keeping all simulation state on the GPU.

This trajectory supports a broader research agenda: enabling *computationally-intensive narrative simulations* that would otherwise be architecturally inaccessible. Games like *Dwarf Fortress* demonstrate the expressive potential of deep simulation; this work asks whether modern data infrastructure can systematically expand what is simulable.

Repository: https://github.com/myouza/columnar-game-logic