Draft Game Design Document

Codename: DataCraft

Roy [roy.ma9@gmail.com]

April 26, 2025

Contents

1 Game Overview

Summary: A factory/simulation-genre game where you design, build, and optimize pipelines to manipulate immutable data units to satisfy business goals.

Genre: Factory, Base Builder, Puzzle, Automation Sim, Optimization-Puzzles

Target Audience: Anyone interested in programming, production-line building, logical thinking, systems optimization puzzles, and engineering in general(e.g., *Factorio, Satisfactory*).

2 Core Gameplay Loop

1. Receive a real-world adjacent business objective (e.g., deliver cleaned customer data to an endpoint within 10 seconds of run, and under $50 overall compute costs).
2. Build data pipelines using various data tools (with functionalities grounded in real-world production-grade tools), following a high-level pipeline: extraction → ingestion → transformation → aggregation → deployment.
3. Watch individual data units (cubes) flow through the system, changing and combining into the final product.
4. Tweak pipeline structure, tools, storage, and compute resources to optimize cost, performance, reliability, and governance. (*Iterative tuning example: start from basic manual data manipulation → rearrange flow → upgrade tool → integrate more pipelines.*)
5. Complete objectives and unlock new tools, more compute resources, budgets, human capital, or advanced challenges.

3 Key Data Concepts

3.1 Immutable Raw Data Cubes

- Core units: each block represents a row of IRL data.
- Individual data blocks cannot be altered once extracted.
- Raw data appear and will disappear immediately unless collected. (or disappear on a timer to not stress players too much)
- Collection methods: batch (on player defined schedules) or streaming (data arrival immediate trigger collection action).
- Note: IRL, raw data must be immutable, otherwise any conclusions drawn from the data to be false.

3.2 Data Types

- Structured: Information arranged as rows/columns (tables).
- Semi-Structured: Nested structures (JSON, XML).
- Unstructured: Binary, images, audio, video, documents.
- Metadata: Timestamps, sources, file types, and sizes. Collected automatically along extractions.

3.3 Storage Systems ("Landing Zones")

- Data Warehouse: Structured, reliable, fast to query, but expensive, and schema-rigid.
- Data Lake: Flexible, cheap, handles any type, but slow to query, needs governance overhead.
- Lakehouse: Hybrid combining warehouse querying speed with lake flexibility. Consider it the trade-off approach between warehouses and lakes.

3.4 Data Ingestion ("Extraction")

- Batch: Scheduled collections and computations. "Collect all of today's customer records by midnight, then execute compute pipeline" (Cheapest, but slow to refresh)

- Streaming: New data triggers immediate collection, computation delayed until reaching mini-batch threshold. "Customer clicks button, event gets ingested immediately to storage, but compute pipeline only execute after N such data have came in" (Fast, but more expensive than batch)
- Real-Time: New data triggers immediate ingestion and compute pipeline execution (most expensive but fastest).

3.5 Data Transformation Tool Examples

- SQL: fundamental data manipulation language, similar to pointing and clicking on excel spreadsheets, but with codes. In game, we can simulate it as basic, single action data manipulations, but can be combined to do very complex manipulations.
- dbt: this is the tool that blueprints SQL actions to do complex, sophisticated tasks. Similar to how players would use blueprints to lay an entire train track system in *factorio*.
- Spark: SQLs are just codes, it needs an engine to run it when you hit "run codes", Spark is the industry standard SQL engine that execute SQL codes cheap, fast, and reliably.

3.6 Workflow Orchestration

- Modern tech is all about automations, actions combines into modules, modules combines into pipelines, pipelines combines into workflows, and workflows are the essential "gameplay loop" of a business unit within any company.
- In the data engineering world, we call our workflow directed acyclic graphs (DAGs). a DAG can be considered as a step-by-step instruction on how and when to execute an action.
- In game, we should have detailed UIs for DAGs, for example: zoom-in, display micro details such as pipes and machineries, zoom-out, display real-time animated UI/graph that shows the DAGs. (other zoom levels may include concepts like job clusters or task groups)
- DAG UI should include Catalog Explorer, Schema Registry, and Versioning (Git).
- The DAG UI is where the macro planning should happen.

3.7 Downstream Consumption Endpoints

- Since we are current focusing on the data engineering angle, the goals of each level is to produce data that satisfy the needs of other data roles (data analyst, data science, or machine learning).
- **For Data Analysts:** Provide cleverly aggregated data so analysts can visualize into beautiful data dashboards that tells a story or business narrative to non-technical stakeholders (e.g., the investors, C-suites).
- **For Data Scientists:** They are the solution identifiers and builders, so they need data that pinpoints problems, then data to feed their machine learning solutions. Data engineers' job is to provide these types of data.
- Machine learning engineers help data scientists build more advanced machine learning solutions.

3.8 Failure Modes

- Each level of the game should pass or not, base on the following criteria:
- Schema Mismatch: Output data missing required columns.
- Invalid Types: Wrong data types in output data.
- Latency Budget Exceeded: Computation exceeds allowed runtime.
- Budget Exceeded: Exceeds allowed compute/storage fiscal cost.
- Downstream Dependency Failure: Poor output data quality breaks downstream processes.
- Silent Data Drift (Advanced): Gradual degradation of raw data quality requiring redesign of data pipelines.

4 Progression & Objectives

- Levels introduce tools incrementally. Tools essentially encapsulates work, better tools enables the player to trade money for faster, more efficient, and better quality output data.
- Cost of tools can be subscription-based or usage-based.

- Levels should be inspired by real-world business challenges.
- Levels should have clear goals and constraints (money, time, tools).
- Endgame involves some type of endless mode where the pipeline can get very complex, close to how the real world works:
  - Fault-tolerant data systems
  - Self-healing data pipelines
  - Machine Learning solutions integrated
  - Cost/Performance trade-offs optimized

5 Visual & UX Design

5.1 Visual Style

- Block-based voxel or clean isometric 2.5D, or similar to shapez.io
- Real-time flowing cubes with tooltips.
- Minimalistic, white-box futuristic, clean sci-fi aesthetic.

5.2 UI Goals

- Modularity and Stackability.
- Deep inspect/debug panels with lots of details (cost, latency, metrics).
- Immediate visual feedback (e.g., cost flow animations, "-$$$" popups).
- Full freedom to manipulate every aspects and details.
- Layered zooms: Action → Module → Pipeline → Workflow.

6 Design Philosophy

- Visual intuitions to scratch the itch of designing and optimizing logical systems
- Complexity as a sandbox design space, enjoy reducing complexity into defined structures
- Design actions to be modular, traceable, testable, and stackable, so it doesn't feel too much like busy work
- Focus on immediate action-reward feedback and system optimization
- Encourage creative sandboxing, allowing players to invent unexpected new solutions

7 Long-Term Visions

- Support community-made logic blueprints and shareable builds via workshops.
- Support for custom-made level constraints and goals.
- Extendable DLCs for Data Science, Machine Learning, AI operations perspectives.
- Optional real-world live data feeds into the game world.

Thank you for reading!

I genuinely believe these ideas have the potential to turn into a great and commercially successful game. The background is rooted in real-world practices that are both ubiquitous and relatable by people who can pay (developers and engineers!), and factory-genre games have consistently proven to perform well in the market (sometimes I relax by playing factory games). So with your team's specific experiences in this genre, these ideas could be brought to life in an amazing way!

As for myself, I'm AI-native, have been coding for 7+ years, and I also have some experience with Unreal Engine. If you decide to further explore these concepts, I'd love to help — whether through technical development, creative design, or wherever support is needed.

Let me know what you think — we all love the chance to work on something we love!

Best regards,
Roy
roy.ma9@gmail.com
April 26, 2025