Tower Defense Data Processing System

Final After-Action Report and System Manual

Commander Eyüb YILDIRIM

July 11, 2025

Mission Synopsis: This document details the design, architecture, deployment, and operational procedures for the Tower Defense Data Processing System. It serves as the definitive guide for system operators and future developers.

Contents

1 System Architecture and Design Philosophy

The Tower Defense System is architected as a modern, real-time, three-tier web application. It simulates a high-throughput data processing pipeline, providing a tactical interface for a human operator to manage and optimize the flow of data "units."

1.1 Technology Stack

The chosen technologies prioritize performance, real-time communication, and a reactive user experience.

- **Backend Command Center: Go (Golang)** was selected for its exceptional performance, first-class support for concurrency via goroutines and channels, and its strong typing system. It forms the backbone of the game engine.
- **Frontend Battle Interface: Vue 3 with TypeScript** provides a highly reactive, component-based architecture. The Composition API is used extensively for clean, reusable logic. **Tailwind CSS** enables rapid, utility-first styling.
- **Real-time Communication System: WebSockets** are used for instantaneous, low-latency, bidirectional communication between the backend engine and the frontend interface.

1.2 Core Design Principles

Decoupling The system is logically divided into independent packages: ‘config‘ for loading parameters, ‘game‘ for core logic, and ‘websocket‘ for communication. The frontend is a completely separate application, communicating only via the WebSocket API.

Concurrency The Go backend leverages a multi-ticker game loop within a ‘select‘ statement, allowing for independent, non-blocking operation of unit spawning, TTL checks, and each of the two weapon systems.

State Management The backend maintains a single, authoritative ‘GameState‘ struct. This state is serialized to JSON and broadcast to all clients upon any change. The frontend is a "dumb" client that simply renders the state it receives.

External Configuration All strategic parameters (rates, timings, scores, unit health, weapon power) are externalized into a ‘config.json‘ file, allowing for dynamic adjustment of game balance without recompiling the application.

2 System Deployment and Operation

The entire system is containerized using Docker and orchestrated with Docker Compose for simplified, one-command deployment.

2.1 Prerequisites

- Docker Engine
- Docker Compose
- ‘make‘ (for using the backend Makefile)

2.2 Project Directory Structure

The deployment assumes the following directory structure. The ‘docker-compose.yml‘ file should be placed in a parent directory, such as ~/Documents/.

```
~/maestrohub-case/
|-- docker-compose.yml
|-- backend/
|     `-- cmd/
|     |-- internal/
|     |-- Dockerfile
|     |-- Makefile
|     `-- config.json
`-- frontend/
    `-- src/
    |-- Dockerfile
    |-- package.json
    `-- ... (other frontend files)
```

2.3 Deployment Procedure

1. **Navigate** to the directory containing the ‘docker-compose.yml‘ file.

   ```
   cd ~/maestrohub-case/
   ```

2. **Build and Run** the entire system with a single command.

   ```
   docker compose up --build
   ```

3. **Access the System** by opening a web browser and navigating to:

   http://localhost:5173

2.4 System Management

The ‘Makefile‘ provides convenient commands for managing the backend service.

- To run the backend locally (without Docker): `make run`
- To build the backend binary: `make build`
- To stop all Docker containers: In the directory with ‘docker-compose.yml‘, run `docker compose down`.

3 System Configuration

The entire game’s balance and difficulty are controlled by the `config.json` file located in the root of the backend project. Changes to this file will take effect the next time the backend server is started.

3.1 Configuration File: `config.json`

Below is an example configuration with explanations for each parameter.

```json
{
  "rates": {
    "spawnRateMs": 3000,
    "individualWeaponRateMs": 500,
    "groupWeaponRateMs": 1000,
    "ttlCheckRateMs": 200
  },
  "timing": {
    "battlefieldTtlSec": 20.0,
    "transitTtlSec": 15.0
  },
  "scoring": {
    "points": {
      "soldier": 10,
      "tank": 50,
      "helicopter": 100
    },
    "penalties": {
      "escape": 200,
      "breach": 500
    }
  },
  "units": {
    "hitpoints": {
      "soldier": 20,
      "tank": 80,
      "helicopter": 150
    }
  },
  "weapons": {
    "processingPower": {
      "individual": 10,
      "group": 25
    }
  }
}
```

3.2 Parameter Breakdown

`rates` Controls the speed of all game events, in milliseconds.

- `spawnRateMs`: Time between each new wave of units spawning. (Lower = More units)
- `individualWeaponRateMs`: The cycle time for the individual weapon. (Lower = Faster firing)
- `groupWeaponRateMs`: The cycle time for the group weapon. (Lower = Faster firing)
- `ttlCheckRateMs`: The frequency of the game's internal clock for updating all TTLs. (Lower = Smoother timers)

`timing` Controls all Time-To-Live durations, in seconds.

- `battlefieldTtlSec`: How long a unit can survive on the main battlefield before it ”Escapes.”

- `transitTtlSec`: How long a deployed unit/group can survive in the defense corridor before it "Breaches."

`scoring` Manages the game's economy.

- `points`: The score awarded for destroying each type of unit.
- `penalties`: The score deducted for 'escape' and 'breach' failure events.

`units` Defines the core properties of military units.

- `hitpoints`: The amount of "damage" each unit can sustain before being destroyed.

`weapons` Defines the effectiveness of the Defense Tower.

- `processingPower`: The amount of "damage" each weapon applies per firing cycle.

4 How to Play: Commander's Guide

Your mission is to manage the flow of chaotic data units, organizing and processing them to maximize your score while preventing system overloads.

4.1 The Battlefield

The main panel shows all available "chaotic" units spawning on the battlefield. Each unit has a **Time-to-Live (TTL)**.

- **Goal:** Process units before their TTL reaches zero.
- **Failure (Escape):** If a unit's TTL expires on the battlefield, it "escapes," incurring a score penalty.

4.2 Actions

You, the Commander, take action using the panels on the left.

1. **Select Units:** Click on one or more units on the battlefield to select them.
2. **Create Squad Blueprint:** With units selected, click this button to save their formation as a reusable "Squad" in the Saved Squads panel. This is a strategic action for long-term planning.
3. **Deploy Individuals:** Send all selected units to the fast, but low-power, "Individual Weapon" queue.
4. **Deploy Squad:** Click "Deploy" next to a saved squad blueprint. The system will intelligently grab all available units that match the blueprint and send them as a single group to the powerful "Group Weapon" queue.

4.3 The Defense Corridor

The right-hand panel provides a real-time visualization of the processing pipeline.

- Deployed units and groups appear at the top and travel towards the Defense Tower at the bottom.
- Each deployed asset has a **Transit TTL**. It must be destroyed before this timer expires.
- **Failure (Border Breach):** If a unit or group's Transit TTL expires because the weapon systems are too busy, it "breaches the border," incurring a severe score penalty.

4.4 Weapon Systems

The ‘WeaponStatus‘ panel shows the real-time state of your processing weapons.

- Each weapon has a certain **Processing Power (PP)**, representing damage per cycle.
- Units have **Hitpoints (HP)**. A weapon must apply damage over several cycles to destroy a unit.
- A weapon is **BUSY** while processing a target and cannot acquire a new one.

4.5 Winning Strategy

Victory is achieved by balancing immediate threats with efficient batch processing.

- Use the Individual Weapon to quickly eliminate low-HP units or high-priority targets that are close to escaping.
- Be wary of tying up a weapon on a high-HP target, as this can cause a backlog and lead to Border Breaches.
- Use the Group Weapon to process large numbers of units efficiently.
- Save effective squad compositions as blueprints to streamline your deployment strategy.
- **Your score is your measure of success. Your goal is to maximize it.**

— END OF REPORT —