Technical Reference Manual

Version: v3.0.1
Date: 2026-01-15

This document provides detailed technical reference material for the current implementation: architecture, module responsibilities, local data models, telemetry, analytics, and the adaptive AI system (Flow Index + contextual bandit).

1. Runtime Architecture

He Maumahara is a pure-frontend system:

- UI: static HTML pages + CSS
- Logic: vanilla JavaScript (browser runtime)
- AI: client-side fuzzy logic scoring and contextual bandit decision layer
- Analytics: local session history and on-device clustering
- Storage: IndexedDB + localStorage

There is no backend service and no network dependency for core features.

2. Codebase Map (Files and Responsibilities)

2.1 Pages

- index.html: home menu
- play.html: level selection
- lvl-1.html / lvl-2.html / lvl-3.html: gameplay pages
- analytics.html: analytics dashboard
- instructions.html / credits.html: informational pages

2.2 JavaScript Modules

File	Responsibility
js/game-core.js	Shared game utilities, telemetry integration, export/screenshot helpers, AI toggle UI glue, AI profile persistence
js/lvl1.js	Level 1 gameplay logic (baseline fixed layout)
js/lvl2.js	Level 2 gameplay logic (adjacency-based layout generation, progression logic)
js/lvl3.js	Level 3 gameplay logic (image–text matching and normalization)
js/ai-helper.js	Telemetry metric extraction and end-of-game AI orchestration
js/ai-engine.js	Flow Index (fuzzy logic), contextual bandit (LinUCB), configuration generation
js/game-history.js	game_history IndexedDB access layer
js/analytics-summary.js	Post-game summary rendering and K-Means overall review

3. Local Persistence Model

3.1 IndexedDB Databases (Conceptual)

Telemetry databases (per level)

- telemetry_lvl1
- telemetry_lvl2
- telemetry_lvl3

History database

- game_history

AI profile database

- ai_player_profile

3.2 localStorage Keys (Conceptual)

- ai_adaptive_enabled: boolean flag for whether adaptation is applied
- ai_level1_config / ai_level2_config / ai_level3_config: serialized “next config” payloads
- lvl2_next_action: local action/state used by Level 2 progression logic

4. Telemetry Event Model

Telemetry events are recorded during play to support analytics and the adaptive loop. Events are stored locally in IndexedDB and can be exported as JSON.

4.1 Common Event Types

Typical events include:

- start: session begins (includes level and a configuration snapshot)
- flip: each card flip
- match: success/failure outcomes for a pair attempt
- show_cards: hint usage (state transitions show/hide)
- ripple_effect: hint animation triggered after repeated errors
- end: session completion summary
- flow_index: Flow Index computation payload and supporting metrics
- ai_suggestion: next configuration suggestion

4.2 Session Segmentation (How a “game” is isolated)

At analysis time, a session is typically defined as:

- the most recent start event for a level, and
- the subsequent end event (if present), with all events in between treated as the current session.

This segmentation enables consistent metric extraction even if the store contains multiple historical runs.

5. Metric Extraction (ai-helper.js)

The AI helper extracts performance signals from telemetry:

- completion time (derived from event timestamps)
- total matches and failed matches
- click count / click efficiency
- flip intervals (cadence series) and cadence stability measures
- hint usage (Show count)
- consecutive error streaks
- optional attribute statistics (color/shape) when image metadata is present

The extracted metrics are the input to Flow Index scoring and configuration selection.

6. Flow Index (Fuzzy Logic Scoring)

6.1 Purpose

The Flow Index compresses multi-signal performance into a single interpretable value in [0, 1].

6.2 Inputs (Representative)

- normalized completion time
- error rate
- click accuracy proxy
- cadence stability proxy
- hint usage penalty

- optional attribute sensitivity proxies

6.3 Output Interpretation

Higher Flow Index indicates better performance under the current configuration. The system aims to keep difficulty appropriate over time rather than maximizing raw difficulty.

User-facing display may clamp values to avoid discouraging feedback while preserving an internal learning signal.

7. Contextual Bandit (LinUCB) and Configuration Selection

7.1 Why a Bandit

A contextual bandit:

- learns online from local rewards (Flow Index)
- selects among discrete actions (difficulty arms)
- balances exploration vs exploitation without centralized training

7.2 Arms

Three arms are used:

- Arm 0: easiest
- Arm 1: standard
- Arm 2: challenge

7.3 Configuration Outputs (Examples)

Depending on level and selected arm, configuration parameters may include:

- gridCols/gridRows for Levels 2 and 3 (5×4 vs 4×6)
- totalPairs derived from grid size
- hideDelay and showScale derived from a hidden difficulty level
- Level 2 adjacency target (assistance) adjusted based on recent Flow Index
- step smoothing to avoid abrupt jumps between arms and grid sizes

8. Analytics

8.1 Post-Game Summary (analytics-summary.js)

The game-over summary panel renders:

- Flow Index and interpretation
- time/accuracy proxies and error indicators
- hint usage
- configuration snapshot (e.g., hideDelay/showScale, adjacency stats on Level 2)

8.2 History (game_history)

Completed sessions are stored to drive analytics.html. A session record typically contains:

- timestamp
- level
- derived metrics/summary fields
- AI results (Flow Index, suggested next configuration)

8.3 K-Means Overall Review (analytics-summary.js)

The overall review clusters recent sessions using a compact feature vector such as:

- Flow Index
- Accuracy
- optional Speed score derived from time-per-pair

Implementation notes:

- min–max normalization stabilizes clustering across different play styles

- K is chosen based on available sample size (e.g., K=2 or K=3)
- outputs include assignments, centroids, and a short trend visual

9. Testing and Tooling

Automated tests under tests/ simulate players against level scripts to provide regression coverage for:

- core gameplay progression under different player profiles
- Flow Index variability under different behavioral patterns
- configuration selection stability

10. Debugging and Inspection

Browser DevTools:

- Application → IndexedDB: telemetry, game_history, ai_player_profile
- Application → Local Storage: ai_* keys and per-level next config

Export:

- gameplay pages provide JSON export for offline analysis and reproducible evaluation.