He Maumahara — Project Brief (Group Meeting Handout)

- Version: v3.0.1 (codebase on disk)
- Date: 2026-01-15
- Audience: Users and Supervisor

This document is a printable, formal overview of the He Maumahara project. It describes what the user experiences, how gameplay works, and how the client-side adaptive AI and analytics operate without any backend services.

1) Executive Summary

He Maumahara is a browser-based memory game designed around culturally grounded Māori imagery and bilingual learning goals. The project emphasizes accessibility for Kaumātua and whānau (high contrast, large touch targets, minimal clutter) and a strict privacy-first architecture.

The technical differentiator is a fully local adaptation loop:

- Telemetry is recorded on-device (IndexedDB).
- A Flow Index (0–1) is computed at the end of each round using a lightweight fuzzy-logic scorer.
- A contextual bandit (LinUCB) selects the next difficulty configuration (three discrete "arms").
- The next configuration is persisted locally (localStorage), so future rounds start at a personalized difficulty.

No backend server is required. No gameplay data is uploaded.

2) User-Facing Experience (Pages and Navigation)

Entry points and routing are implemented as static HTML pages with standard navigation links.

Main Menu

- File: index.html
- Actions: Play, Analytics, Instructions, Credits, Quit

Level Selection

- File: play.html
- Actions: Level 1, Level 2, Level 3

Gameplay Pages (one page per level)

- Files: lvl-1.html, lvl-2.html, lvl-3.html
- Common controls:
    - Menu: returns to play.html
    - Show: temporarily reveals non-matched cards (time penalty)
    - Export: downloads telemetry events (JSON)
    - Adapt: toggles adaptive difficulty (persisted in localStorage)
    - Reset: clears the level telemetry database

Game-Over Panel (within each level page)

- Shows elapsed time (mm:ss) and a post-game analytics summary.
- Actions: Screenshot, Download (screenshot + JSON), Next Game (progression using the latest AI config).

Analytics Dashboard

- File: analytics.html
- Modes:
    - History: reads saved sessions from IndexedDB (game_history)
    - Demo: uses mock data to illustrate the charts
- Includes a K-Means "Overall Review" to summarize recent performance patterns.

3) Gameplay Model (Core Loop and Level Differences)

Core loop shared across all levels:

- Initialize and render a card grid.
- Run a short initial preview where cards are visible, then hide them and start the round timer.
- Accept card clicks; compare two flipped cards at a time.
- On match: mark as matched, increment progress, and apply a small time reward (capped).
- On mismatch: flip back after a delay; repeated errors can trigger a brief ripple hint animation.
- End the round when all pairs are matched or when the timer reaches zero.

Local telemetry is recorded throughout the session to support analytics and adaptation.

Level 1 (Visual Baseline: Image–Image)

- Fixed 5×4 grid and fixed card order to reduce randomness for onboarding.
- Time cost for Show is lower than later levels.

Level 2 (Spatial Challenge: Image–Image)

- Grid size can be 5×4 or 4×6 (selected by the AI configuration).
- Layout generation includes adjacency assistance: a target number of pairs may be placed adjacent to reduce search difficulty for struggling players.
- Level progression logic can either keep the player on Level 2 (with a modified grid) or advance to Level 3.

Level 3 (Linguistic Challenge: Image–Text)

- Each pair consists of an image card and a matching text card (kupu).
- Matching uses normalization so image keys and text labels map reliably.
- Adds semantic recall on top of visual memory.

4) Adaptive AI (Local, Privacy-First Personalization)

The adaptive system is a small "edge AI" pipeline that runs entirely in the browser and learns per user.

Key concepts (short technical introduction)

- Fuzzy logic: converts multiple noisy signals (time, errors, cadence, hint usage) into a stable score without requiring a heavy model.
- Contextual bandit (LinUCB): an online learning method that chooses among discrete actions while balancing exploitation (what works) and exploration (what might work better).

Implementation modules

- js/ai-helper.js: extracts metrics from telemetry and orchestrates end-of-game processing.
- js/ai-engine.js: computes Flow Index and decides the next configuration.

Data signals used by the AI (examples)

- Completion time, total matches, failed matches, and click counts.
- Flip interval stability (cadence).
- Hint usage (Show) and consecutive error streaks.
- Optional image attribute statistics (color/shape) when available.

Decision outputs (what changes between rounds)

- Difficulty "arm" (three arms: easy, standard, challenge).
- For Levels 2 and 3: grid size (5×4 or 4×6) with step-smoothing to avoid sudden jumps.
- Hidden difficulty parameters that control reveal dynamics (hideDelay and showScale).
- For Level 2: adjacency assistance targets that are adjusted gradually based on recent Flow Index.

Persistence (device-local only)

- Adaptive toggle: localStorage (ai_adaptive_enabled)
- Next configuration per level: localStorage (ai_level1_config, ai_level2_config, ai_level3_config)
- Long-term history: IndexedDB (game_history)
- Telemetry stream: IndexedDB per level (telemetry_lvl1, telemetry_lvl2, telemetry_lvl3)

5) Analytics (Local Progress Visualisation)

Post-Game Summary

- Rendered on the game-over screen to provide immediate feedback.
- Includes Flow Index, completion time, error indicators, hint usage, and a snapshot of the configuration used.

Analytics Dashboard

- Reads completed sessions from IndexedDB (game_history).
- Provides per-session summaries and an overall "recent performance" view.

K-Means Overall Review (unsupervised clustering)

- Purpose: compress many recent games into a small number of performance patterns.
- Features: Flow Index, Accuracy, and an optional Speed score derived from time-per-pair.
- Output: a Flow vs Accuracy scatter plot, a dominant cluster label, and a short trend strip.

6) Privacy and Data Ownership

- All telemetry, AI state, and history remain on the user's device.
- Export is user-initiated (JSON download), enabling offline evaluation without any server component.
- The architecture aligns with "data sovereignty by default": there is no remote collection surface to secure or govern.

7) Testing and Demonstration Notes

Suggested meeting demo flow (5–8 minutes)

1. index.html → play.html → Level 1: show the preview phase and play a few pairs.
2. Use Show once to demonstrate the time penalty and explain "hint usage" as an input signal.
3. Finish the round: highlight the post-game summary, Flow Index, and the "Next Game" flow.
4. Proceed to Level 2: point out layout variation and adjacency assistance.
5. Open analytics.html: show History and the K-Means Overall Review.

Automated simulation tests are available under tests/ (Node-based) to regression-test behavior.

8) File Map (For Q&A)

Pages

- index.html (home)
- play.html (level selection)
- lvl-1.html / lvl-2.html / lvl-3.html (gameplay)
- analytics.html (dashboard)
- instructions.html / credits.html (information)

Core logic

- js/game-core.js (telemetry, shared utilities, export/screenshot, history integration)
- js/lvl1.js (Level 1 baseline logic)
- js/lvl2.js (Level 2 layout generation and progression)
- js/lvl3.js (Level 3 image-text matching)

AI and analytics

- js/ai-helper.js (metric extraction and end-of-game AI orchestration)
- js/ai-engine.js (Flow Index + contextual bandit decision logic)
- js/game-history.js (IndexedDB session storage)
- js/analytics-summary.js (post-game summary and K-Means overall review)