Ear Fatigue / Auto-Dynamics Plugin - Agile Development Plan

Goal: To create an MVP audio plugin that provides visual feedback on dynamic range reduction, helping producers maintain objectivity and avoid over-compression due to ear fatigue or chasing perceived loudness.

Context: Solo developer, beginner in C++/JUCE, learning alongside development (approx. 1 -2 hours/day).

(Phase checklist to be updated as project progresses)

Phase 1: Conceptualization & Planning (Partially Complete)

- ☑ Define the plugin's core functionality and target users (Producers, esp. less experienced).
- ☑ Establish the unique selling proposition (Visual feedback on actual dynamic reduction vs. perceived loudness; guidance on listening level adjustment).
- ☑ Identify core MVP features:
    - ☑ Dynamic range visualization (Traffic light).
    - ☑ Algorithm detecting dynamics reduction (compression/limiting amount).
    - ☑ Warning indicator for excessive reduction.
    - ☑ Simple UI (Bypass, Standard Selection).
    - ☐ Visual representation/guidance for user listening volume adjustment (Define method - e.g., text message).
- ☐ Create basic User Stories based on MVP pain points (e.g., "As a producer, I want to see when I'm reducing dynamics too much, so I don't create flat mixes.") and prioritize.
- ☑ Choose appropriate DSP algorithms (Initial thoughts: RMS/Peak detection, potentially LUFS basics for context, core algorithm TBD based on research/learning).
- ☑ Select frameworks (JUCE) and target formats (Initial: VST3, AU).
- ☑ Acknowledge Knowledge Gaps (Advanced psychoacoustics, specific maths for core algorithm, Fletcher-Munson nuances). Plan learning alongside development.

Phase 2: Foundational Learning & Basic Prototype Setup (Aligns with Learning Schedule "Learn First" & "JUCE Project Setup")

- [ ] **Learn C++ Fundamentals:** Complete core C++ learning modules (Syntax, OOP, Memory Management/RAII, STL).
- [ ] **Learn Real-Time Audio Concepts:** Understand audio callbacks, block processing, sample rates, buffer sizes, and crucial thread safety rules (no allocations, locks etc. in processBlock).
- [ ] **Setup JUCE Environment:** Install JUCE & Projucer.
- [ ] **Create Basic JUCE Plugin Project:** Generate the initial plugin template using Projucer.
- [ ] **Build & Run Empty Plugin:** Ensure the template plugin builds successfully and loads in at least one target DAW.
- [ ] **Understand Core JUCE Classes:** Familiarize with juce::AudioProcessor and juce::AudioProcessorEditor structure and lifecycle (prepareToPlay, processBlock, releaseResources, paint, resized).
- [ ] **Implement Basic Audio Pass-through/Bypass:** Get audio flowing through the processBlock and implement a functional bypass mechanism.
- [ ] **Establish Parameter Handling:** Implement AudioProcessorValueTreeState and add the first parameter (e.g., Bypass toggle).

Phase 3: Core DSP Implementation Sprints (Aligns with Learning Schedule "Intermediate" DSP)

- *(Iterative Sprints - ~1-2 weeks each, focused)*

- [ ] **Sprint 3.1: Basic Level Detection:** Implement reliable Peak and RMS calculation within processBlock.
- [ ] **Sprint 3.2: Research & Design Core Algorithm:** Define the mathematical/logical approach to estimate dynamic range reduction/compression based on audio analysis (beyond simple LUFS). Document the chosen approach. *This is a critical research + design step.*
- [ ] **Sprint 3.3: Implement Core Algorithm (v1):** Code the initial version of the dynamics reduction detection algorithm. Output results via logging or debugging.
- [ ] **Sprint 3.4: Refine & Test Algorithm:** Test with various audio material. Refine the algorithm based on observations. Does it react plausibly to compressed vs. dynamic audio?

- [ ] **Sprint 3.5: Parameter Integration:** Add parameters needed for the algorithm (e.g., potentially threshold, reference level if user-settable in future) via APVTS.

Phase 4: Minimal Viable UI & Visualization (Aligns with Learning Schedule "Then" & "Intermediate" GUI/Graphics)

- [ ] **Implement Basic GUI Structure:** Create the AudioProcessorEditor, link parameters (Bypass) to basic JUCE GUI components (juce::ToggleButton, SliderAttachment, etc.).
- [ ] **Develop Traffic Light Visualization:**
    - [ ] Use juce::Graphics or simple components to draw the traffic light indicators.
    - [ ] Use a juce::Timer to periodically get the latest DSP results.
    - [ ] Map the core algorithm's output value to the traffic light states (Green/Amber/Red) based on defined thresholds (e.g., link to Mastering Standard selection).
- [ ] **Implement Standard Selection:** Add UI element (e.g., juce::ComboBox) to select different mastering standards/references, linking it to a parameter and adjusting algorithm thresholds accordingly.
- [ ] **Implement Listening Guidance Display:** Add the defined visual element (e.g., juce: :Label showing text like "Consider lowering listening volume" or "Dynamics heavily reduced") triggered by algorithm state.

Phase 5: Integration, MVP Feature Completion & Basic Testing (Aligns with Agile Phase 4 & 6 - Simplified)

- [ ] **Integrate all Components:** Ensure DSP, parameters, and UI work together smoothly.
- [ ] **Basic Functionality Testing:**
    - [ ] Test in target DAWs (your primary ones).
    - [ ] Use varied audio material (dynamic, compressed, different genres).
    - [ ] Verify Bypass works correctly.
    - [ ] Verify standard selection changes behavior as expected.
    - [ ] Verify traffic light and guidance messages correspond logically to the input audio dynamics.

- [ ] **Basic Performance Check:** Monitor CPU usage in the DAW – ensure it's reasonable and doesn't cause dropouts.
- [ ] **Identify & Fix Critical Bugs:** Address crashes, major functional errors, or audio artifacts.

-- MVP Complete Milestone --

(Subsequent Phases - Post-MVP Focus)

Phase 6: Alpha Testing / Refinement (Self-Testing Focus)

- [ ] Extended testing with personal music projects.
- [ ] Usability review: Is it intuitive? Does it help achieve the goal?
- [ ] Refine algorithm thresholds and behavior based on real-world use.
- [ ] Minor UI polish.

Phase 7: Beta Release & Iteration (Optional / If pursuing wider use)

- [ ] Share with trusted testers.
- [ ] Gather feedback.
- [ ] terate based on feedback (focus on stability, usability, core algorithm accuracy).

Phase 8: Documentation & Presets (Minimal initially)

- [ ] Write basic usage instructions/tooltips.
- [ ] Define default settings/initial preset.

Phase 9: Release & Continuous Improvement (Future)

- [ ] Build release versions.
- [ ] Monitor for issues.
- [ ] Plan future enhancements based on learning and potential feedback.

This tailored plan integrates the necessary learning steps directly into the development

phases. It prioritizes getting the core DSP logic working (Phase 3) as that's the unique heart of your plugin, even before the UI is fully fleshed out. We can use this document going forward to track progress. How does this look to you?