AI統合DAW 段階的開発ガイド

プロジェクト名: Composer Copilot
対象読者: Cursor AI & 開発者
バージョン: 1.0.0
最終更新日: 2025年6月19日

📋 開発概要

このガイドは、AI統合DAWの開発を段階的に進めるためのステップバイステップ手順書です。各ステップには明確なタグが付けられており、Cursor AIが段階を把握しやすくなっています。

プロジェクト構成

```
ai-daw/
├── src/
│   ├── audio/          # オーディオエンジン
│   ├── ui/             # ユーザーインターフェース
│   ├── ai/             # AI統合機能
│   ├── midi/           # MIDI処理
│   └── utils/          # ユーティリティ
├── tests/              # テストコード
├── docs/               # ドキュメント
├── assets/             # リソースファイル
└── CMakeLists.txt      # ビルド設定
```

🚀 Phase 1: プロジェクト基盤構築

// STEP 1: プロジェクト初期化

目標: 基本的なJUCEプロジェクト構造を作成

実装内容: - CMakeLists.txt設定 - 基本的なディレクトリ構造作成 - JUCEとTracktion Engineのサブモジュール追加 - 基本的なMainComponent作成

成果物: - `CMakeLists.txt` - プロジェクトビルド設定 - `src/MainComponent.h/.cpp` - メインアプリケーションウィンドウ - `src/Main.cpp` - アプリケーションエントリーポイント

検証方法: - プロジェクトがビルドできること - 空のウィンドウが表示されること

// STEP 2: 基本UI構造作成

目標: DAWの基本的なレイアウトを実装

実装内容: - メニューバー作成 - ツールバー作成 - トラック表示エリア作成 - タイムライン表示エリア作成 - ステータスバー作成

成果物: - `src/ui/MenuBarComponent.h/.cpp` - メニューバー - `src/ui/ToolbarComponent.h/.cpp` - ツールバー - `src/ui/TrackAreaComponent.h/.cpp` - トラック表示エリア - `src/ui/TimelineComponent.h/.cpp` - タイムライン - `src/ui/StatusBarComponent.h/.cpp` - ステータスバー

検証方法: - 各UIコンポーネントが正しく表示されること - レイアウトがリサイズに対応していること

// STEP 3: オーディオエンジン基盤

目標: Tracktion Engineを統合した基本的なオーディオエンジンを実装

実装内容: - AudioEngine クラス作成 - Tracktion Engine初期化 - 基本的なEdit（プロジェクト）管理 - オーディオデバイス設定

成果物: - `src/audio/AudioEngine.h/.cpp` - メインオーディオエンジン - `src/audio/ProjectManager.h/.cpp` - プロジェクト管理 - `src/audio/DeviceManager.h/.cpp` - オーディオデバイス管理

検証方法: - オーディオデバイスが正しく認識されること - 新しいプロジェクトが作成できること - 基本的なオーディオ再生ができること

🎵 Phase 2: 基本DAW機能実装

// STEP 4: トランスポートコントロール

目標: 再生、停止、録音などの基本的なトランスポート機能を実装

実装内容: - TransportControls クラス作成 - 再生/停止/録音ボタン - テンポ設定 - 拍子設定 - ループ機能

成果物: - `src/ui/TransportControls.h/.cpp` - トランスポートUI - `src/audio/TransportManager.h/.cpp` - トランスポート制御ロジック

検証方法: - 再生/停止が正しく動作すること - テンポ変更が反映されること - ループ機能が動作すること

// STEP 5: トラック管理システム

目標: オーディオトラックとMIDIトラックの基本的な管理機能を実装

実装内容: - Track クラス階層設計 - AudioTrack と MidiTrack 実装 - トラック追加/削除機能 - トラックミュート/ソロ機能 - ボリューム/パン制御

成果物: - `src/audio/Track.h/.cpp` - 基底トラッククラス - `src/audio/AudioTrack.h/.cpp` - オーディオトラック - `src/audio/MidiTrack.h/.cpp` - MIDIトラック - `src/ui/TrackComponent.h/.cpp` - トラックUI

検証方法: - トラックの追加/削除ができること - ミュート/ソロが正しく動作すること - ボリューム調整が反映されること

// STEP 6: MIDI基本機能

目標: MIDI入力、編集、再生の基本機能を実装

実装内容: - MIDIInputManager 作成 - MIDISequence 管理 - ピアノロールエディタ基本版 - MIDIノート編集機能

成果物: - `src/midi/MidiInputManager.h/.cpp` - MIDI入力管理 - `src/midi/MidiSequence.h/.cpp` - MIDIシーケンス - `src/ui/PianoRollEditor.h/.cpp` - ピアノロールエディタ - `src/midi/MidiNote.h/.cpp` - MIDIノートデータ

検証方法: - MIDI入力が正しく認識されること - ピアノロールでノートが表示されること - ノートの編集ができること

🤖 Phase 3: AI統合機能実装

// STEP 7: MCP通信基盤

目標: Model Context Protocol (MCP) を使用したAI通信基盤を実装

実装内容: - MCPClient クラス作成 - 非同期通信機能 - エラーハンドリング - 複数AIプロバイダー対応

成果物: - `src/ai/MCPClient.h/.cpp` - MCPクライアント - `src/ai/AIProvider.h/.cpp` - AIプロバイダー抽象クラス - `src/ai/ClaudeProvider.h/.cpp` - Claude統合 - `src/ai/ChatGPTProvider.h/.cpp` - ChatGPT統合

検証方法: - AI APIとの接続が確立できること - 基本的なリクエスト/レスポンスが動作すること - エラー時の適切な処理ができること

// STEP 8: Agent機能実装

目標: 自然言語からMIDIパターンを生成するAgent機能を実装

実装内容: - AgentProcessor クラス作成 - プロンプト入力UI - MIDI生成ロジック - 生成結果のプレビュー機能

成果物: - `src/ai/AgentProcessor.h/.cpp` - Agent処理エンジン - `src/ui/AgentPanel.h/.cpp` - Agent操作UI - `src/ai/PromptEngine.h/.cpp` - プロンプト処理 - `src/ai/MidiGenerator.h/.cpp` - MIDI生成

検証方法: - 自然言語プロンプトからMIDIが生成されること - 生成されたMIDIが再生できること - 生成結果をトラックに追加できること

// STEP 9: Ghost Text基盤

目標: リアルタイムMIDI予測のための基盤を実装

実装内容: - GhostTextEngine クラス作成 - Python統合機能 - リアルタイム予測パイプライン - 予測結果表示UI

成果物: - `src/ai/GhostTextEngine.h/.cpp` - Ghost Text処理エンジン - `src/ai/PythonBridge.h/.cpp` - Python統合 - `src/ui/GhostTextDisplay.h/.cpp` - 予測結果表示 - `python/ghost_text_model.py` - Python予測モデル

検証方法: - MIDI入力に対してリアルタイム予測ができること - 予測結果が視覚的に表示されること - 予測を受け入れて入力できること

💰 Phase 4: 有料化システム実装

// STEP 10: ライセンス管理システム

目標: フリーミアムモデルのライセンス管理機能を実装

実装内容: - LicenseManager クラス作成 - ローカルライセンス検証 - サーバー認証機能 - 機能制限管理

成果物: - `src/license/LicenseManager.h/.cpp` - ライセンス管理 - `src/license/FeatureGate.h/.cpp` - 機能制限 - `src/ui/LicenseDialog.h/.cpp` - ライセンス管理UI - `src/network/AuthClient.h/.cpp` - 認証クライアント

検証方法: - ライセンス状態が正しく判定されること - 有料機能が適切に制限されること - ライセンス購入フローが動作すること

// STEP 11: 使用量制限システム

目標: AI機能の使用量制限とトラッキングを実装

実装内容: - UsageTracker クラス作成 - API使用量カウント - 制限値管理 - 使用状況表示

成果物: - `src/license/UsageTracker.h/.cpp` - 使用量追跡 - `src/ui/UsageDisplay.h/.cpp` - 使用状況表示 - `src/license/QuotaManager.h/.cpp` - 制限管理

検証方法: - AI機能の使用量が正しくカウントされること - 制限に達した時に適切に制限されること - 使用状況が正確に表示されること

🎨 Phase 5: UI/UX改善

// STEP 12: モダンUI実装

目標: プロフェッショナルなDAWらしい洗練されたUIを実装

実装内容: - カスタムLookAndFeel作成 - ダークテーマ実装 - アニメーション効果 - レスポンシブデザイン

成果物: - `src/ui/ModernLookAndFeel.h/.cpp` - カスタムテーマ - `src/ui/AnimationManager.h/.cpp` - アニメーション管理 - `src/ui/ThemeManager.h/.cpp` - テーマ管理

検証方法: - UIが統一されたデザインになっていること - アニメーションがスムーズに動作すること - 異なる画面サイズに対応していること

// STEP 13: キーボードショートカット

目標: 効率的な操作のためのキーボードショートカットを実装

実装内容: - KeyboardShortcutManager 作成 - カスタマイズ可能なショートカット - ショートカット設定UI - コンテキスト依存ショートカット

成果物: - `src/ui/KeyboardShortcutManager.h/.cpp` - ショートカット管理 - `src/ui/ShortcutSettingsDialog.h/.cpp` - 設定UI

検証方法: - 主要な操作がキーボードで実行できること - ショートカットのカスタマイズができること - コンテキストに応じて適切なショートカットが有効になること

🔧 Phase 6: 最適化とテスト

// STEP 14: パフォーマンス最適化

目標: リアルタイムオーディオ処理に適したパフォーマンス最適化を実装

実装内容: - オーディオスレッド最適化 - メモリ使用量最適化 - CPU使用率監視 - プロファイリング機能

成果物: - `src/utils/PerformanceMonitor.h/.cpp` - パフォーマンス監視 - `src/audio/OptimizedAudioProcessor.h/.cpp` - 最適化されたオーディオ処理

検証方法: - レイテンシが許容範囲内であること - CPU使用率が適切であること - メモリリークがないこと

// STEP 15: 包括的テスト実装

目標: 品質保証のための包括的なテストスイートを実装

実装内容: - 単体テスト作成 - 統合テスト作成 - UI自動テスト - パフォーマンステスト

成果物: - `tests/unit/` - 単体テストスイート - `tests/integration/` - 統合テストスイート - `tests/performance/` - パフォーマンステスト

検証方法: - すべてのテストがパスすること - テストカバレッジが80%以上であること - CI/CDパイプラインが正常に動作すること

📦 Phase 7: 配布準備

// STEP 16: インストーラー作成

目標: Windows用のプロフェッショナルなインストーラーを作成

実装内容: - WiX Toolsetを使用したMSIインストーラー - デジタル署名 - 自動更新機能 - アンインストール機能

成果物: - `installer/` - インストーラー設定ファイル - `scripts/build_installer.bat` - インストーラービルドスクリプト

検証方法: - インストーラーが正常に動作すること - アプリケーションが正しくインストールされること - アンインストールが完全に実行されること

// STEP 17: ドキュメント整備

目標: ユーザー向けドキュメントとヘルプシステムを整備

実装内容: - ユーザーマニュアル作成 - インアプリヘルプシステム - チュートリアル機能 - FAQ作成

成果物: - `docs/user_manual.md` - ユーザーマニュアル - `src/ui/HelpSystem.h/.cpp` - ヘルプシステム - `docs/tutorials/` - チュートリアル

検証方法: - ドキュメントが分かりやすいこと - ヘルプシステムが正常に動作すること - チュートリアルが効果的であること

🚀 Phase 8: リリース準備

// STEP 18: ベータテスト準備

目標: ベータテスト用のビルドとフィードバック収集システムを準備

実装内容: - ベータ版ビルド設定 - クラッシュレポート機能 - フィードバック収集UI - テレメトリー機能

成果物: - `src/telemetry/CrashReporter.h/.cpp` - クラッシュレポート - `src/ui/FeedbackDialog.h/.cpp` - フィードバック収集

検証方法: - ベータ版が正常にビルドできること - クラッシュレポートが正しく送信されること - フィードバックが収集できること

// STEP 19: 最終品質保証

目標: リリース前の最終的な品質保証を実施

実装内容: - 全機能の動作確認 - パフォーマンステスト - セキュリティ監査 - 互換性テスト

成果物: - `tests/release/` - リリーステストスイート - `docs/qa_checklist.md` - QAチェックリスト

検証方法: - すべての機能が仕様通りに動作すること - パフォーマンス要件を満たしていること - セキュリティ要件を満たしていること

// STEP 20: リリース実行

目標: 製品の正式リリースを実行

実装内容: - リリースビルド作成 - 配布サイト準備 - マーケティング材料準備 - サポート体制構築

成果物: - 最終リリースビルド - 配布用Webサイト - プレスリリース - サポートドキュメント

検証方法: - リリースビルドが正常に動作すること - 配布チャネルが準備できていること - サポート体制が整っていること

📝 開発時の注意事項

コード品質

- 各ステップで適切なテストを作成する
- コードレビューを必ず実施する
- ドキュメントを同時に更新する
- パフォーマンスを常に意識する

AI統合

- API制限を考慮した実装を行う
- エラーハンドリングを徹底する
- ユーザー体験を最優先に考える
- セキュリティを十分に検討する

プロジェクト管理

- 各ステップの完了基準を明確にする
- 定期的な進捗確認を行う
- 問題が発生した場合は早期に対応する
- ユーザーフィードバックを積極的に収集する

注意: このガイドは開発の進行状況に応じて更新される可能性があります。各ステップの実装前に最新版を確認してください。