更新版 修正計画書

1. 背景とゴール

1.1 背景

- **タスク一覧のクリック操作**と、**チャットでタスク名を直接入力**する操作の処理フローが異なっているため、実際に同一タスクを実行するはずが、
  - 例：UIで「支給品控除等一覧処理」をクリックした場合と、チャットで同様の指示を出した場合とで、内部のコード分岐や状態管理（AgentCollection内の current_task の設定や状態遷移）がずれており、
  - ファイルアップロード手順がスキップされたり、意図しないステートマシンの遷移が発生するなど、ユーザーに混乱とエラーをもたらしています。
- **ユーザーが自分でルール（.mdファイル）を作成するフロー**について、現状は開発者が md や json を手作業で作成しているため、
  - 業務担当者が GUI 操作（チャット＋テンプレート）を用いてルール作成、テスト、確定できる仕組みが求められています。
- **クラス構造（AgentCollection、TaskAgent など）**が大きく肥大化しており、
  - UI操作や新機能追加のフックが増えるにつれて保守性が低下しているため、今後の拡張（ルール作成画面、タスク設定画面など）に耐えられるよう、責務の分割と再整理が必要です。

1.2 ゴール

1. **タスク一覧クリックでもチャット入力でも同じタスク処理が実行できる**ようにし、どちらの操作でもエラーなく処理が流れる統一的なフローを実現する。
2. **ルール作成のUIボタンを追加し、ユーザーがチャット＋テンプレートを用いてルールを作成、テスト、保存できるフロー**をサポートする。
3. **クラス設計を見直し、責務を明確に分割して保守性・拡張性を向上**させる。
   - 例：ルール作成機能は専用の `RuleCreationAgent`（仮称）、config.jsonの読み書きは `ConfigManager`（仮称）、タスク実行は `TaskAgent`、エージェント全体の統合管理は `AgentCollection` とする。

---

2. 問題の詳細と改善の方向性

2.1 タスク一覧クリックとチャット入力で処理が異なる問題

- **問題点**
  - UIクリック時の状態遷移（たとえば、AgentCollection 内で current_task の設定や状態マシンの移行処理）と、チャットで「支給品控除等一覧処理」などの意図を伝えた際の状態遷移が統一されていないため、
  - 結果として「ファイルアップロード手順がスキップされる」「意図しないステートマシンの遷移が起こる」などのエラーが発生している。
- **改善策**
  1. **タスク起動フローの一元化**

    - タスク一覧のクリック操作もチャットでの指示も、最終的には同じエントリーポイント（例：`AgentCollection.set_current_task_agent(task_name)`）を経由して処理が開始されるように変更する。
    - UI側のクリックイベントは内部的に「タスク選択」というチャットインテントに対応させ、同一の状態遷移および初期化処理を行う。
2. **エラーハンドリングの統合**
    - 必要ファイルがアップロードされていない場合や日付指定が未完了の場合など、各ケースで共通のエラーメッセージおよび操作ガイドを返すよう統一する。

2.2 ルール作成フローのユーザーフレンドリー化

- **問題点**
    - ユーザーがチャットを通じて「こういうルールを作りたい」と入力しても、現状は開発者が直接 .md ファイルや json を作成する必要があるため、
    - ルールのテスト実行・保存のプロセスが手動で行われ、ミスが生じやすい状況になっている。
- **改善策**
    1. **タスク一覧画面に「ルール作成」ボタンを追加**
        - ボタンを押すと、どのタスクに紐付けるかや、新規ルールか既存タスクに対するルールかを選択するダイアログまたは別画面を表示する。
    2. **対話型ウィザード or チャットUIの導入**
        - ユーザーに対して「必要な列は何ですか？」「どのような計算ロジックを実装したいですか？」などの質問を順次行い、その回答を基に Markdown 形式のルールテンプレートを生成、プレビュー表示する。
    3. **テスト実行と自動保存機能**
        - 生成したルールを一時的にメモリに保持し、ユーザーがサンプルファイルでテスト実行を行い、問題なければ「保存」ボタンを押すと自動的に .md ファイルと config.json に反映される仕組みにする。

2.3 クラス設計の再整理

- **問題点**
    - `AgentCollection` や `TaskAgent` が多くの機能を内包するようになっており、UI操作や新規機能追加のフックが増えるほどに保守が困難となっている。
- **改善策**
    1. **層の分離**
        - UI層（View＋Controller）、ドメイン層（タスク状態管理、ルール生成）、インフラ層（DB操作、ファイル操作）の3層構造を意識して各責務を分割する。
    2. **新規モジュールの設置**
        - 例：ルール作成機能を担当する `RuleCreationAgent`（仮称）を新設し、チャット入力をもとに Markdown テンプレートの生成・プレビュー処理を一手に引き受ける。
        - config.json の読み書きは `ConfigManager`（仮称）に切り出し、タスク実行ロジックは `TaskAgent` に専念させ、エージェント全体の統合管理は `AgentCollection` が行うようにする。
    3. **依存関係の整理**
        - 各クラス間の依存性を見直し、単体テストや統合テストを導入して、今後の機能追加時にも後方互換を保てるようにする。

---

3. プログラム修正の具体的施策

ここからは、上記の改善策を現行のプログラム各ファイル（agent_collection.py、task_agent.py、main.py など）に対してどのように反映させるか、具体的な修正方針を示します。

3.1 AgentCollection（agent_collection.py）の修正

- **タスク起動フローの統一**
  - UIのタスク一覧クリック時のイベントハンドラ（main.py 内の _on_cell_click など）から呼び出される際、内部では必ず `set_current_task_agent(task_name)` を利用してタスクを開始する。
  - 現在、チャット入力での意図解析結果に基づく処理（`IntentType.TASK_START` の場合の `parse_task_names` など）と、UIクリック時の処理を同一の初期化・状態遷移メソッドに集約する。
- **状態遷移・エラーハンドリングの統合**
  - 各意図（ファイルアップロード、確認応答など）ごとに、エラーが発生した場合のメッセージや、ユーザーへ次の操作を促すメッセージを統一する。
  - 具体的には、`process_message()` や各 _process_XXX メソッドのエラーメッセージ部分を共通化し、UIクリック時とチャット入力時の双方で同じ状態マシンの遷移が実現できるように修正する。

3.2 ルール作成機能のUI追加および連携（新規モジュールの導入）

- **新規クラス RuleCreationAgent（仮称）の実装**
  - チャットやフォーム入力により、ユーザーからルール作成に必要な情報（例：必要な列、計算ロジック、例外処理）を順次収集する対話型ウィザードを実装。
  - 入力情報をもとに Markdown 形式のルールテンプレートを自動生成し、画面上にプレビューを表示する。
  - テスト実行を行い、問題がなければ最終的に .md ファイルおよび config.json（もしくは該当タスクの設定）にルールを保存する機能を実装する。
- **UI側の修正（main.py）**
  - タスク一覧画面またはチャット画面に「ルール作成」ボタンを追加する。
  - ボタン押下時にルール作成用のダイアログ（または別ウィンドウ）を起動し、RuleCreationAgent のフローに乗せる。
  - ダイアログ内では、入力フォーム、プレビューエリア、テスト実行ボタン、保存ボタンなどを実装し、ユーザーが直感的に操作できるようにする。

3.3 TaskAgent（task_agent.py）の修正

- **処理ステップの見直しと統一**
  - タスクの各ステップ（prepare、file、analysis、process、test、work、revise、continue、over）の状態遷移が、UI操作とチャット入力の双方で一貫して動作するように、
    - fast_mode と通常モードの処理分岐の整理を行う。
  - ファイルチェックやコード生成、テスト実行において、ルール作成で生成されたテンプレート（prompt_content）の利用方法を統一し、
    - 必要に応じてルール編集機能（RuleCreationAgent との連携）に処理を委譲する。
- **ルール編集機能の分離**

    - タスク実行に必要なルールの読み込みは、TaskAgent 内ではなく RuleCreationAgent や ConfigManager に委譲することで、TaskAgent の責務を「タスク実行」に専念させる。

3.4 ConfigManager の新設（config_manager.py の作成）

- **config.json の読み書きの責務を分離**
    - 現在、AgentCollection 内で行われている設定ファイル（config.json）の読み込み処理を ConfigManager クラスに移行する。
    - ConfigManager は、タスク設定の取得、タスクの追加・更新、ルール保存時の設定反映などを担当し、AgentCollection は ConfigManager を通じて必要な設定情報を参照するようにする。

3.5 その他のモジュールの修正

- **IntentAnalyzer（intent_analyzer.py）**
    - ルール作成やタスク選択に関連する意図解析において、UI操作とチャット入力の両方を適切に判断できるよう、プロンプト内容や解析結果のフォーマットを微調整する。
- **FileAgent（file_agent.py）**
    - 各タスクの必要ファイルのアップロードチェック、ファイル内容の一致確認などの処理が、UI操作（ファイル選択ダイアログ経由）とチャット操作の双方で同一の動作となるように見直す。
- **コードエージェント（code_agent.py）・チャットエージェント（chat_agent.py）**
    - ルール生成のためのテンプレートやチェーンの更新処理において、ユーザーがルール作成UIで入力した内容が正しく反映されるよう、パラメータの受け渡しと後処理（例："import pandas as pd" の強制追加など）を確認・調整する。

3.6 UI の修正（main.py 等）

- **タスク一覧クリックの統一処理**
    - タスク一覧のセルクリックイベント（_on_cell_click）では、選択されたタスクグループ内の各タスク名を AgentCollection の workflow に順次追加し、
        - その後、必ず `set_current_task_agent(task_name)` を呼び出してタスク開始の一元化された処理経路に乗せる。
- **ルール作成ボタンの追加**
    - 画面上に「ルール作成」ボタンを配置し、クリック時にルール作成用ダイアログを起動する。
    - ダイアログでは、ユーザーが必要な項目（ファイルの列名、計算ロジック、例外処理など）を入力でき、プレビュー、テスト実行、保存の各操作を実施できるようにする。
- **ユーザーフィードバックの改善**
    - エラー発生時のメッセージや、次のステップへの誘導メッセージを統一し、UI・チャットともにユーザーが迷わず操作できるようなガイドラインを実装する。

---

4. 開発スケジュール（段階的リリース計画）

4.1 フェーズ1：タスク実行フローの統一修正

- **対象**: AgentCollection、TaskAgent、UIのタスク一覧クリックイベント
- **作業**:
    1. UIクリック時とチャット入力時の処理を統一し、必ず `set_current_task_agent(task_name)` を経由するようにリファクタリングする。

    2. 状態マシンの状態遷移（chat、file、date、task など）を整理し、共通の初期化およびエラーハンドリング処理を実装する。
- **期間目安**: 1〜2週間
- **テスト**: タスク一覧クリック→ファイルアップロード→対話（Yes/No入力）→結果取得の一連の流れで動作確認

4.2 フェーズ2：ルール作成UIとチャット連携の実装

- **対象**: 新規 RuleCreationAgent、UI（main.py ）、ルール作成ダイアログ
- **作業**:
    1. タスク一覧画面に「ルール作成」ボタンを追加し、ダイアログ表示を実装する。
    2. ルール作成ウィザード（対話形式 or フォーム形式）を構築し、ユーザー入力から Markdown テンプレートを生成する。
    3. テスト実行機能を実装し、問題がなければ「保存」ボタンで .md および config.json へ自動反映する処理を組み込む。
- **期間目安**: 2〜3週間
- **テスト**: サンプルファイルでのテスト実行、エラー処理、最終保存の一連の流れを検証

4.3 フェーズ3：クラス設計の再整理・リファクタリング

- **対象**: AgentCollection、TaskAgent、ConfigManager、RuleCreationAgent など
- **作業**:
    1. 各クラスの責務分割と依存関係を見直し、UI層、ドメイン層、インフラ層への分離を実施する。
    2. 単体テスト・統合テストを整備し、今後の拡張時にも堅牢な設計とする。
- **期間目安**: 並行して実施または約2週間
- **成果**: 保守性・拡張性の向上、後方互換性の確保

4.4 段階的リリース

1. **フェーズ1完了後**: 統一されたタスク実行フローをリリースし、ユーザーがエラーなくタスク処理できることを確認。
2. **フェーズ2完了後**: ルール作成UI機能をリリースし、ユーザーが自律的にルールを編集・テスト・保存できる仕組みを提供。
3. **フェーズ3完了後**: クラス設計の再整理をリリースし、システム全体の保守性と拡張性を確保する。

---

5. まとめ

本修正計画では、以下の点に重点を置いてプログラムの改修を進めます：

- **タスク一覧クリック時とチャット入力時の処理差異の解消**
  UI操作とチャット入力の両方から同一のタスク起動処理
  （`AgentCollection.set_current_task_agent()` 経由）を呼び出し、状態遷移とエラーハンドリングを統一する。

- **ルール作成フローのユーザーフレンドリー化**
  ルール作成ボタンと対話型ウィザード／フォームを用いたルール作成UIを実装し、ユーザーがチャット入力だけでなく直感的なGUI操作によりルール作成、テスト、保存を実施できる仕組みを提供する。

- **クラス設計の再整理と責務分離**
  AgentCollection、TaskAgent、そして新規の ConfigManager や RuleCreationAgent を導入し、各クラスの役割を明確化することで、今後の機能追加や改修コストの低減、システムの拡張性を確保する。

これらの修正により、ユーザーは直感的にタスクの選択と実行ができ、さらに業務担当者が自律的に新たなルールを作成・テスト・保存できるアプリケーションへと進化します。
また、内部のクラス構造の再整理により、今後の追加機能や改修作業もスムーズに行える堅牢な設計となります。