프로젝트 리팩토링 계획

이 문서는 EVE Abyssal Tracker 프로젝트의 코드베이스를 리팩토링하기 위한 상세 계획을 담고 있습니다. 목표는 코드 가독성 및 유지보수성 향상, 객체 지향 설계 원칙 준수, 모듈화 및 구조 개선입니다.

1. 리팩토링 목표

1. **코드 가독성 및 유지보수성 향상:**
    - 불필요하거나 중복된 설명용 주석, 또는 코드의 의미를 명확히 전달하지 못하는 주석을 제거합니다. 코드가 스스로 설명하도록 개선합니다.
    - 사용되지 않는 코드(데드 코드, 미사용 변수/함수 등)를 식별하고 제거하여 코드베이스를 간결하게 유지합니다.
    - 변수 및 함수명을 의미를 명확히 전달하도록 개선합니다.
2. **객체 지향 설계 원칙 준수:**
    - 클래스 및 모듈이 단일 책임 원칙(SRP)을 준수하고 있는지 확인하고, 응집도를 높이고 결합도를 낮추는 방향으로 재설계합니다.
    - 캡슐화, 상속, 다형성 등 객체 지향의 핵심 원칙이 적절히 적용되었는지 검토하고 개선합니다.
3. **모듈화 및 구조 개선:**
    - 지나치게 길거나 여러 책임을 가진 파일을 논리적인 단위(클래스, 함수, 모듈)로 분리하여 코드의 재사용성과 관리 용이성을 높입니다.
    - 명확한 책임과 인터페이스를 가진 모듈 구조를 확립합니다.
4. **로직 변경 시 사용자 확인:**
    - 기존 비즈니스 로직의 변경이 필요하다고 판단되는 경우, 반드시 사용자에게 변경의 필요성, 예상되는 영향, 그리고 대안을 명확히 설명하고 승인을 받은 후에 진행합니다.

2. 상세 리팩토링 계획

A. 초기 분석 및 준비 (완료)

- 전체 파일 구조 파악
- 주요 모듈의 코드 정의 파악
- 각 파일의 역할 및 잠재적 리팩토링 대상 식별

B. 상세 분석 및 리팩토링 (모듈별)

1. **`src/logic/abyssal_data_manager.py` 및 `src/logic/global_abyssal_data_manager.py` 통합 및 재설계:**
    - **목표:** 두 클래스의 책임 범위를 명확히 정의하고, 중복되는 기능을 제거하여 SRP를 강화합니다.
    - **세부 계획:**

        - `AbyssalDataManager`는 개별 어비셜 런 데이터의 저장 및 로드, 파싱을 담당하도록 재정의합니다.
        - `GlobalAbyssalDataManager`는 모든 런 데이터를 집계하고 통계 분석을 수행하는 역할을 하도록 재정의합니다.
        - 필요시 `data_processing_utils.py`와 같은 새로운 모듈을 생성하여 공통 데이터 처리 로직을 분리합니다.
    - **사용자 확인:** 이 변경은 데이터 관리 방식에 영향을 미치므로, 변경 전에 사용자에게 상세 내용을 설명하고 승인을 받겠습니다.

2. `src/logic/tracker.py` **모듈 분리 및 SRP 강화:**
    - **목표:** `AbyssalRunTracker` 클래스의 여러 책임을 분리하여 응집도를 높이고 결합도를 낮춥니다.
    - **세부 계획:**
        - 로그 파일 모니터링 (`LogMonitor` 또는 `LogWatcher` 클래스)
        - 시스템 변경 감지 및 파싱 (`SystemChangeDetector` 또는 `SystemLogParser` 클래스)
        - 팝업 관리 (`PopupManager` 또는 `UINotifier` 클래스)
        - 각각의 새로운 클래스는 명확한 단일 책임을 가지도록 설계합니다.
    - **Mermaid 다이어그램:**

```
graph TD
    A[AbyssalRunTracker (기존)] --> B{책임 분리};
    B --> C[LogMonitor];
    B --> D[SystemChangeDetector];
    B --> E[PopupManager];
    C --> F[로그 파일 읽기];
    D --> G[시스템 변경 감지];
    E --> H[UI 팝업 관리];
    F -- 사용 --> D;
    G -- 알림 --> E;
```

3. `src/logic/eve_log.py` **개선:**
    - **목표:** 로그 처리 로직의 모듈화 및 관리 용이성 향상.
    - **세부 계획:**
        - 로그 언어 감지 및 패턴 매칭 로직을 더 모듈화합니다.
        - 정규 표현식 패턴을 별도의 설정 파일이나 상수 모듈로 분리하여 관리 용이성을 높입니다.

4. `src/logic/eve_api.py` **개선:**
    - **목표:** API 캐싱 로직 견고화 및 에러 핸들링 강화.
    - **세부 계획:**
        - API 캐싱 로직을 더 견고하게 만들고, 에러 핸들링을 강화합니다.
        - API 호출 관련 로직과 데이터 처리 로직을 분리할 수 있는지 검토합니다.

5. `src/ui/ui_popup.py` **개선 (및** `src/ui/ui_stats_display.py` **제거):**

- **목표:** `src/ui/ui_stats_display.py`는 사용되지 않으므로 제거하고, `src/ui/ui_popup.py`의 UI 클래스 응집도를 향상시킵니다.
- **세부 계획:**
  - `src/ui/ui_stats_display.py` 파일 제거.
  - `src/ui/ui_popup.py`에서 UI 로직과 데이터 처리 로직이 혼재되어 있다면 분리하여 UI 클래스의 응집도를 높입니다.
  - Tkinter 위젯 생성 및 배치 로직을 더 깔끔하게 정리합니다.

6. `streamlit_pages/stats_display.py` **개선:**

- **목표:** Streamlit 앱의 역할 명확화 및 백엔드 로직 위임.
- **세부 계획:**
  - `app()` 함수가 너무 길다면, 내부 로직을 더 작은 함수나 클래스로 분리합니다.
  - 데이터 로딩 및 분석 로직을 `GlobalAbyssalDataManager`와 같은 백엔드 로직으로 위임하여 Streamlit 앱은 순수하게 UI 렌더링에만 집중하도록 합니다.

C. 전반적인 코드 품질 향상

- **주석 및 데드 코드 제거:** 각 파일을 리팩토링하면서 불필요한 주석과 데드 코드를 제거합니다. 특히, 사용되지 않는 `src/ui/ui_stats_display.py` 파일을 제거했습니다.
- **변수/함수명 개선:** 가독성을 높이기 위해 변수 및 함수명을 더 명확하게 변경합니다.
- **에러 핸들링:** 기존 `try-except` 블록을 검토하고, 필요한 경우 더 구체적인 예외 처리 및 로깅을 추가합니다.
- **타입 힌트:** Python 3.5+ 환경이라면 타입 힌트를 추가하여 코드의 명확성을 높입니다.

D. 테스트 및 검증

- 리팩토링 후 각 기능이 정상적으로 작동하는지 확인합니다. (수동 테스트 또는 자동화된 테스트 코드 작성 고려)

E. 문서화

- 주요 변경 사항 및 새로운 모듈 구조에 대한 문서를 업데이트합니다.