제공해주신 기획서(Master GDD)와 코드베이스 분석 보고서, 그리고 주요 파일(turnLogic.js, Timeline.jsx, useBattleState.js)을 바탕으로 **[Star Seeker] 전투 시스템 구현을 위한 단계별 개발 계획**을 수립했습니다.

이 계획은 \*\*'기존 리액트 기반 UI + 로직 중심 설계'\*\*를 따르며, GDD의 핵심인 \*\*"거리 기반 타임라인"\*\*과 \*\*"유저 개입(CP 스킬)"\*\*을 구현하는 데 초점을 맞췄습니다. 개발자가 Copilot에게 즉시 지시할 수 있도록 구체적인 프롬프트 포인트와 작업 단위를 나누었습니다.

# 🚀 [Star Seeker] 전투 시스템 개발 마스터 플랜

## 🛠 선행 작업: 코드 베이스 정리 (Cleanup)

CODEBASE\_ANALYSIS\_REPORT.md에서 지적된 대로, 비대한 useGameLogic.js에서 전투 로직을 분리하는 것이 최우선입니다.

* **지시 사항:**
  1. useGameLogic.js에서 전투 관련 상태(battleStatus, party, enemy 등)를 제거하고 useBattleSystem 훅으로 이관.
  2. src/utils/battle/turnLogic.js의 에러(throw new Error)를 제거하고 빈 함수로 초기화하여 로직 작성 준비.

## 📅 Phase 1: 전투 데이터 구조 및 엔진 설계 (Core Engine)

**목표:** GDD 7장 '핵심 로직 명세서'에 따라 유닛의 스탯, 거리, 턴 계산 로직을 확립합니다.

### 1-1. 유닛 데이터 모델링 (Unit Schema)

GDD 7.3 및 2.1을 기반으로 전투에 참여하는 객체(Unit)의 표준 모델을 정의합니다.

* **작업 위치:** src/constants/battle/types.js (신규 생성 또는 기존 파일 보강)
* **Copilot 지시 포인트:**

"전투 유닛 객체(BattleUnit)의 타입을 정의해줘.

* 1. 기본 스탯: HP, MaxHP, Atk, Def, Spd, Crit, Hit, Res.
  2. 전투 자원: EP(0~100), 거리(Distance: 0~10000).
  3. 식별자: uid, isEnemy, element(속성), role(직업군).
  4. distance의 초기값은 10000(Start), 목표값은 0(Action)이야."

### 1-2. 타임라인 및 턴 연산 로직 구현

src/utils/battle/turnLogic.js를 GDD 7.1에 맞춰 재작성합니다.

* **작업 위치:** src/utils/battle/turnLogic.js
* **핵심 로직:**
  + **거리 공식:** $Pos\_{Current} = Pos\_{Prev} - Speed$
  + **Next Actor 산출:** 가장 빨리 0에 도달하는 유닛을 찾고, 그 시간($t$)만큼 모든 유닛을 전진시킴.
* **Copilot 지시 포인트:**

"calculateNextState(units) 함수를 구현해줘.

* 1. 모든 생존 유닛의 (현재 거리 / 속도)를 계산해서 '도착까지 남은 시간(MinTime)'이 가장 짧은 유닛을 찾아.
  2. 모든 유닛의 거리를 현재 거리 - (속도 \* MinTime)으로 업데이트해.
  3. 거리가 0 이하가 된 유닛을 nextActor로 반환해.
  4. 단, 이미 행동권이 있는(거리가 0인) 유닛은 제외하고 계산해야 해."

## 🌌 Phase 2: 타임라인 UI 고도화 (Visualization)

**목표:** Timeline.jsx를 GDD 2.2 '별자리 궤도' 컨셉에 맞춰 완성하고 실제 데이터와 연동합니다.

### 2-1. 타임라인 컴포넌트 완성

현재 Timeline.jsx는 SVG 궤도만 있습니다. 유닛 아이콘의 동적 배치와 애니메이션을 보강합니다.

* **작업 위치:** src/components/battle/Timeline.jsx
* **Copilot 지시 포인트:**

"Timeline.jsx를 수정해서 유닛들이 궤도 위를 부드럽게 이동하도록 해줘.

* 1. unit.distance (0~10000)를 SVG 경로상의 좌표 (x, y)로 변환하는 로직은 이미 있어.
  2. 유닛 아이콘(div absolute)에 transition: all 0.5s linear를 적용해서 위치 변경 시 부드럽게 미끄러지듯 보이게 해줘.
  3. 적군은 붉은색 테두리/그림자, 아군은 푸른색 테두리/그림자로 구분해줘(GDD 2.2)."

## 🃏 Phase 3: 유저 개입 시스템 (Intervention & CP)

**목표:** GDD 3.2 및 7.1에 명시된 '시간 조작 스킬'을 구현합니다. 이 게임의 핵심 재미 요소입니다.

### 3-1. 개입(Intervention) 로직 함수 구현

유닛의 distance를 직접 조작하는 함수들을 만듭니다.

* **작업 위치:** src/utils/battle/interventionLogic.js (신규)
* **핵심 기능:**
  + 🧲 **Pull (당기기):** distance = 1 (즉시 턴 획득 직전 상태로)
  + ✋ **Push (밀기):** distance += 5000 (최대 10000 제한)
  + 🔄 **Swap (교차):** 두 유닛의 distance 값 맞교환
* **Copilot 지시 포인트:**

"전투 유닛의 위치를 강제로 변경하는 함수들을 작성해줘.

* 1. applyPull(unit): 유닛의 distance를 1로 설정.
  2. applyPush(unit): 유닛의 distance에 5000을 더함 (Max 10000).
  3. applySwap(unitA, unitB): 두 유닛의 distance 값을 서로 바꿈."

### 3-2. CP 자원 관리 및 스킬 발동 UI

전투 화면 하단에 CP 스킬 버튼을 배치하고 로직을 연결합니다.

* **작업 위치:** src/components/battle/BattleControls.jsx
* **Copilot 지시 포인트:**

"하단 패널에 '개입 스킬' 버튼 4개를 배치해줘. (아이콘: 🧲, ✋, 🔄, ⚫)

* 1. 각 버튼 클릭 시 타겟팅 모드(아군 선택/적군 선택)로 진입해야 해.
  2. CP가 부족하면 버튼을 비활성화해줘. (당기기/밀기: 1, 교차: 2, 궁극기: 4)
  3. 타겟 선택이 완료되면 3-1에서 만든 로직 함수를 실행하고 CP를 차감해줘."

## ⚔️ Phase 4: 전투 실행 및 결과 처리 (Action Resolution)

**목표:** GDD 4장 및 6장에 따른 공격, 스킬, 속성 상성 처리를 구현합니다.

### 4-1. 데미지 및 상성 공식 적용

GDD 6장(속성)과 7.3(스탯 공식)을 반영한 계산기를 만듭니다.

* **작업 위치:** src/utils/battle/battleCalculator.js
* **Copilot 지시 포인트:**

"calculateDamage(attacker, defender, skillMultiplier) 함수를 만들어줘.

* 1. 속성 상성 로직(GDD 6장)을 적용해. (우위 속성일 때 데미지 +30%).
  2. 치명타 확률(crit)에 따라 데미지 1.5배 적용.
  3. 최종 데미지 = 공격력 \* 스킬계수 \* 상성보정 \* 방어력감쇄."

### 4-2. 턴 종료 및 리셋 (Turn End)

행동을 마친 유닛을 다시 10,000 거리로 돌려보내는 로직입니다.

* **작업 위치:** src/hooks/useBattleSystem.js
* **Copilot 지시 포인트:**

"유닛이 행동(공격/스킬)을 완료하면 실행할 onActionComplete 함수를 작성해줘.

* 1. 행동한 유닛의 distance를 10000으로 리셋해 (GDD 2.2 Omega Star).
  2. 아군의 경우 공격 종류에 따라 EP를 회복시켜줘 (평타 +25, 스킬 +30~40).
  3. 그 후 다시 calculateNextState를 호출해서 다음 턴을 진행해."

## 🏗 개발 진행 가이드 (개발자 -> Copilot)

실제 작업을 시작할 때, 아래 순서대로 Copilot에게 프롬프트를 입력하며 파일을 생성/수정해 나가세요.

**Step 1. 유닛 타입 정의 (기반 다지기)**

"GDD 7.3 항목을 참조하여 src/constants/battle/types.js에 BattleUnit 인터페이스와 초기 스탯 계산 로직을 작성해줘."

**Step 2. 턴 로직 구현 (심장 만들기)**

"GDD 7.1의 거리 기반 턴 시스템을 구현하려고 해. src/utils/battle/turnLogic.js에 calculateNextState 함수를 작성해. 속도에 따라 남은 거리를 차감하고, 거리가 0이 되는 유닛을 찾는 로직이야."

**Step 3. 타임라인 시각화 (뼈대 입히기)**

"src/components/battle/Timeline.jsx를 열어줘. allUnits 배열을 받아서 각 유닛의 distance에 따라 SVG 경로 위의 위치를 계산하고, 아이콘을 렌더링하도록 수정해줘."

**Step 4. 개입 스킬 로직 (재미 요소 추가)**

"GDD 3.2의 Pull, Push, Swap 기능을 구현해야 해. src/hooks/useBattleSystem.js 내부에 유저가 CP를 소모해서 특정 유닛의 distance를 즉시 변경하는 핸들러 함수들을 만들어줘."

**Step 5. 전투 흐름 연결 (통합)**

"useBattleSystem 훅에서 턴 계산 -> 유저 행동 선택 -> 데미지 계산 -> 턴 종료(거리 리셋) -> 다시 턴 계산으로 이어지는 useEffect 기반의 게임 루프를 완성해줘."