### **Refactoring Brief: Overhauling the Combat System**

**High-Level Objective:**Your task is to perform a major refactoring of the game's combat system. The goal is to completely remove the existing turn-based UI and mechanics, reducing the combat state to a simple blank screen. This is a "strip-down" operation to create a clean foundation for a new combat system to be built later.

**Project Context:**The current system is a turn-based combat loop with several dedicated UI panels for displaying turn order, enemy info, player stats, and action menus. Player and AI actions are queued and processed sequentially. We are preparing to replace this with a new, different style of combat, so all visual and mechanical elements of the old system must be surgically removed.

**Guiding Principles for this Refactor:**

1. **Surgical Removal:** Only remove code directly related to the old turn-based system. Do not alter non-combat systems like world map movement, the terminal, or the clock.
2. **Preserve the Core State Loop:** The game must still be able to enter and exit the IsInCombat state. The trigger for entering combat (a hostile AI getting close to the player) must remain functional.
3. **Maintain AI Behavior (Pre-Combat):** The AI's ability to detect, chase, and trigger the IsInCombat state must be preserved.
4. **Create a Blank Slate:** The end result of entering combat should be a black screen with minimal text. The game should not crash. It should be possible to exit this state via a debug key.

### **Detailed Breakdown of Changes:**

You will be performing this refactor by deleting a specific list of files and then modifying several others to remove their dependencies and obsolete logic.

**1. Complete Removal of the Combat UI Layer & Input:**The entire UI for the turn-based system will be removed. This involves deleting the following files, which are no longer needed:

* **UI Panels:** ActionMenuPanel.cs, CombatLogPanel.cs, EnemyDisplayPanel.cs, PlayerStatusPanel.cs, TargetInfoPanel.cs, TurnOrderPanel.cs.
* **Input System:** PlayerCombatInputSystem.cs.
* **State Enum:** CombatUIState.cs.

**2. Decoupling the Main Game Scene (TerminalMapScene.cs):**This is the most significant part of the refactor. You will modify this scene to:

* Remove all field declarations and initialization logic for the deleted UI panels.
* In the Update() method, replace the entire if (IsInCombat) block. The new logic will simply check for a debug key (F5) to call \_coreState.EndCombat(). No other combat-related updates should occur.
* In the Draw() method, replace the entire if (IsInCombat) block. The new logic will clear the screen to black and draw the simple text "COMBAT" in the center. This creates the "blank slate" effect.

**3. Neutralizing Combat Mechanics & Turn Progression:**The systems that drove the old combat loop will be hollowed out, leaving only the scaffolding required to cycle turns without performing any real actions.

* **AISystem.cs**: The ProcessCombatTurn() method will be gutted. Instead of planning attacks or moves, it will now simply log a placeholder message and immediately call EndCurrentTurn().
* **CombatTurnSystem.cs**: The StartNewTurn() method will be simplified. All logic related to TurnStatsComponent and status effect processing will be removed. Its only remaining jobs are to advance the turn index, set the CurrentTurnEntityId, and trigger the AI's now-empty turn logic.
* **CombatProcessingSystem.cs**: This system's Update() loop will be emptied. It will no longer dequeue or process any actions (MoveAction, AttackAction, etc.). The ApplyMoveActionEffects method will be deleted.

**4. Stripping Combat-Specific Components & Data Models:**The data structures that supported the old mechanics will be removed.

* **Deleted Components:** TurnStatsComponent.cs, AttackAction.cs, ChosenAttackComponent.cs, EndTurnAction.cs.
* **GameState.cs**: The UIState property and the GetAffordablePath() method will be deleted, as they are tied to the removed components and UI.

**5. Final Logic Cleanup:**The last few dependencies will be severed.

* **CombatResolutionSystem.cs**: This will be reduced to a pass-through. InitiateAttackResolution() and HandleDiceRollCompleted() will be replaced with logic that simply logs a generic "X attacks Y!" message and immediately signals completion, without rolling dice or calculating damage.
* **MapInputHandler.cs**: The "Attack" option will be removed from the right-click context menu on entities.
* **StatusEffectSystem.cs**: All logic related to PoisonStatusEffect and ProcessCombatTurnStart will be removed.

**The Desired End State:**After all modifications, the game should compile and run. The player can navigate the world map as before. When a hostile AI gets close enough, the screen will turn black and display "COMBAT". The game will not crash. The player can then press **F5** to exit combat and return to the normal map view. The AI that triggered combat will still be present and hostile, and may re-engage if the player is still in range.

### **++++++++++++++++++++++++++++++++++++++++++++++++++++++++++**

### **Prompt 1 of 4: Removing Combat UI and Input Systems**

You are a Senior C# Game Developer specializing in MonoGame. I’m working on a MonoGame project and I need your help with the following refactor.

This is the first step in a multi-part process to overhaul the combat system. In this step, we will remove all the UI panels and input systems related to the old turn-based combat.

**I. Deleted Files**

The following files have been deleted from the project. You no longer need to track or modify them:

* ActionMenuPanel.cs
* CombatLogPanel.cs
* EnemyDisplayPanel.cs
* PlayerStatusPanel.cs
* TargetInfoPanel.cs
* TurnOrderPanel.cs
* PlayerCombatInputSystem.cs
* CombatUIState.cs

**II. Files to Modify**

Please apply the following changes to TerminalMapScene.cs:

1. **Remove Fields:** Delete all field declarations for the deleted UI panels listed above.
2. **Modify Enter():** Remove the entire block that initializes the combat UI panels and the PlayerCombatInputSystem.

Instruction Format & Content Requirements

1. Complete, Self‑Contained Files  
   o Only output the source files you modify.  
   o Begin your response with a single, brief introductory sentence like "Here are the updated files:"  
   o Give a brief summary of what was changed. Give instructions on how and where to tune parameters added if tuning said parameters may be desired.  
   o For each file changed, place the file name on its own line, followed immediately by a single fenced code block containing the entire file.  
   o Use the csharp language identifier for the code block (e.g., ```csharp).  
   o Do not leave placeholders like “// rest of code unchanged.”  
   o Make sure to put the title fenced outside of the code block after its output too
2. No Abbreviations or Partial Snippets  
   o Every file you output must compile on its own (with any necessary using‑directives, namespaces, class declarations, etc.).
3. Clarify Ambiguities  
   o If any part of my request is unclear, ask me one concise clarification question.  
   o Offer multiple‑choice options plus an “Other (please describe)” option.
4. Pure Text Instructions  
   o Do not wrap these guidelines or your questions in code blocks.
5. Only Changed Files  
   o Do not reproduce files that require no edits.  
   Whenever you respond, follow these rules exactly so that your outputs are immediately usable in my MonoGame codebase.

### **Prompt 2 of 4: Implementing the Blank Combat Screen**

You are a Senior C# Game Developer specializing in MonoGame. I’m working on a MonoGame project and I need your help with the following refactor.

This is the second step in overhauling the combat system. In this step, we will modify the main game scene's Update and Draw loops to create the "blank screen" combat state.

**Files to Modify**

Please apply the following changes to TerminalMapScene.cs:

1. **Modify Update() method:** Replace the entire if (\_coreState.IsInCombat) block with a new block that only contains a debug keybind (F5) to exit combat.
2. **Modify Draw() method:** Replace the entire if (\_coreState.IsInCombat) block with logic that clears the screen to black and draws the word "COMBAT" in the center.

Instruction Format & Content Requirements

1. Complete, Self‑Contained Files  
   o Only output the source files you modify.  
   o Begin your response with a single, brief introductory sentence like "Here are the updated files:"  
   o Give a brief summary of what was changed. Give instructions on how and where to tune parameters added if tuning said parameters may be desired.  
   o For each file changed, place the file name on its own line, followed immediately by a single fenced code block containing the entire file.  
   o Use the csharp language identifier for the code block (e.g., ```csharp).  
   o Do not leave placeholders like “// rest of code unchanged.”  
   o Make sure to put the title fenced outside of the code block after its output too
2. No Abbreviations or Partial Snippets  
   o Every file you output must compile on its own (with any necessary using‑directives, namespaces, class declarations, etc.).
3. Clarify Ambiguities  
   o If any part of my request is unclear, ask me one concise clarification question.  
   o Offer multiple‑choice options plus an “Other (please describe)” option.
4. Pure Text Instructions  
   o Do not wrap these guidelines or your questions in code blocks.
5. Only Changed Files  
   o Do not reproduce files that require no edits.  
   Whenever you respond, follow these rules exactly so that your outputs are immediately usable in my MonoGame codebase.

### **Prompt 3 of 4: Gutting Combat Logic Systems**

You are a Senior C# Game Developer specializing in MonoGame. I’m working on a MonoGame project and I need your help with the following refactor.

This is the third step in overhauling the combat system. In this step, we will neutralize the core turn-based logic systems, removing their ability to process attacks, moves, and status effects during combat.

**I. Deleted Files**

The following component files have been deleted from the project. You no longer need to track or modify them:

* TurnStatsComponent.cs
* AttackAction.cs
* ChosenAttackComponent.cs
* EndTurnAction.cs

**II. Files to Modify**

Please apply the following changes:

1. **AISystem.cs**: Replace the entire body of the ProcessCombatTurn() method to simply log a message and end the AI's turn.
2. **CombatTurnSystem.cs**: In the StartNewTurn() method, remove all logic related to TurnStatsComponent and status effect processing.
3. **CombatProcessingSystem.cs**: Empty the body of the Update() method (after the initial null checks and IsInCombat guard clause). Also, delete the ApplyMoveActionEffects method entirely.
4. **GameState.cs**: Delete the GetAffordablePath() method, as it relies on the now-deleted TurnStatsComponent.

Instruction Format & Content Requirements

1. Complete, Self‑Contained Files  
   o Only output the source files you modify.  
   o Begin your response with a single, brief introductory sentence like "Here are the updated files:"  
   o Give a brief summary of what was changed. Give instructions on how and where to tune parameters added if tuning said parameters may be desired.  
   o For each file changed, place the file name on its own line, followed immediately by a single fenced code block containing the entire file.  
   o Use the csharp language identifier for the code block (e.g., ```csharp).  
   o Do not leave placeholders like “// rest of code unchanged.”  
   o Make sure to put the title fenced outside of the code block after its output too
2. No Abbreviations or Partial Snippets  
   o Every file you output must compile on its own (with any necessary using‑directives, namespaces, class declarations, etc.).
3. Clarify Ambiguities  
   o If any part of my request is unclear, ask me one concise clarification question.  
   o Offer multiple‑choice options plus an “Other (please describe)” option.
4. Pure Text Instructions  
   o Do not wrap these guidelines or your questions in code blocks.
5. Only Changed Files  
   o Do not reproduce files that require no edits.  
   Whenever you respond, follow these rules exactly so that your outputs are immediately usable in my MonoGame codebase.

### **Prompt 4 of 4: Final Logic and Component Cleanup**

You are a Senior C# Game Developer specializing in MonoGame. I’m working on a MonoGame project and I need your help with the following refactor.

This is the final step in overhauling the combat system. We will remove the last pieces of the old system, including damage calculations, status effects, and the player's ability to initiate combat from the map.

**I. Deleted Files**

The following file has been deleted from the project. You no longer need to track or modify it:

* PoisonStatusEffect.cs

**II. Files to Modify**

Please apply the following changes:

1. **CombatResolutionSystem.cs**:
   * Replace the body of the InitiateAttackResolution() method to immediately call the completion handler.
   * Replace the body of the HandleDiceRollCompleted() method to simply log a generic attack message and complete the action.
2. **MapInputHandler.cs**: In the HandleRightClickOnEntity() method, remove the code block that adds the "Attack" option to the context menu.
3. **StatusEffectSystem.cs**:
   * Remove the "Poison" case from the switch statement in the CreateEffectFromName() method.
   * Empty the body of the ProcessCombatTurnStart() method.

Instruction Format & Content Requirements

1. Complete, Self‑Contained Files  
   o Only output the source files you modify.  
   o Begin your response with a single, brief introductory sentence like "Here are the updated files:"  
   o Give a brief summary of what was changed. Give instructions on how and where to tune parameters added if tuning said parameters may be desired.  
   o For each file changed, place the file name on its own line, followed immediately by a single fenced code block containing the entire file.  
   o Use the csharp language identifier for the code block (e.g., ```csharp).  
   o Do not leave placeholders like “// rest of code unchanged.”  
   o Make sure to put the title fenced outside of the code block after its output too
2. No Abbreviations or Partial Snippets  
   o Every file you output must compile on its own (with any necessary using‑directives, namespaces, class declarations, etc.).
3. Clarify Ambiguities  
   o If any part of my request is unclear, ask me one concise clarification question.  
   o Offer multiple‑choice options plus an “Other (please describe)” option.
4. Pure Text Instructions  
   o Do not wrap these guidelines or your questions in code blocks.
5. Only Changed Files  
   o Do not reproduce files that require no edits.  
   Whenever you respond, follow these rules exactly so that your outputs are immediately usable in my MonoGame codebase.