

Design Document: Combat Damage Indicators

1. Overview

This feature will implement a visual feedback system for combat. When the player attacks a monster, a small text "indicator" will be displayed in the 3D game world near the monster. This text will show the numerical damage dealt, "MISS" for a missed attack, or "NO DAMAGE" for a successful hit that was fully blocked or resulted in zero damage.

This system will be modeled after classic SNES RPGs (e.g., Final Fantasy). The text will appear, float upwards slightly, and fade to full transparency over a short duration (approx. 1 second).

2. Architectural Approach

We will create a self-contained rendering system to manage these temporary text objects. This system will be decoupled from the `CombatManager` (which handles *logic*) and the `MonsterRenderer` (which handles *sprites*).

1. **Data Object (`AttackResult`):** `CombatManager` will be modified to return a simple data object, `AttackResult`, which contains the outcome of an attack (hit status and damage amount).
2. **Data Object (`DamageIndicator`):** A new class, `DamageIndicator`, will be created to store the state of a single piece of floating text (its text, 3D position, lifetime timer, and alpha).
3. **Renderer (`DamageIndicatorRenderer`):** A new renderer class, `DamageIndicatorRenderer`, will manage a list of active `DamageIndicator` objects. It will be responsible for updating their state (position, fade) each frame and rendering them to the screen.
4. **Integration (`GameScreen`):** The `GameScreen` will be the central coordinator. It will:
 - Call `CombatManager` and receive the `AttackResult`.
 - Determine the text to display ("5", "MISS", etc.).
 - Tell the `DamageIndicatorRenderer` to create a new indicator at the monster's 3D position.
 - Call the `DamageIndicatorRenderer`'s `update` and `render` methods in its main game loop.

3. New and Modified Classes

1. `core/src/main/java/com/bpm/minotaur/gamedata/AttackResult.java` (New File)

- **Purpose:** A simple data-transfer object (DTO) to communicate the result of an attack from `CombatManager` to `GameScreen`.
- **Fields:**
 - `public boolean didHit;`
 - `public int damageDealt;`

2. `core/src/main/java/com/bpm/minotaur/rendering/DamageIndicator.java` (New File)

- **Purpose:** Represents a single piece of floating text in the game world.
- **Fields:**
 - `Vector3 position;` (The 3D world position)
 - `String text;`
 - `float lifetime;` (Counts down from `FADE_TIME`)
 - `float alpha;` (Calculated from `lifetime` for fading)
 - `boolean active;` (Set to `false` when `lifetime` hits 0)
 - `static final float FADE_TIME = 1.0f;` (Total duration in seconds)
 - `static final float FLOAT_SPEED = 0.5f;` (World units to float up per second)

3.

`core/src/main/java/com/bpm/minotaur/rendering/DamageIndicatorRenderer.java` (New File)

- **Purpose:** Manages the lifecycle and rendering of all active `DamageIndicator` objects.
- **Key Methods:**
 - `public DamageIndicatorRenderer(BitmapFont font):` Constructor.
 - `public void showIndicator(String text, Vector3 monsterPosition):` Creates and adds a new `DamageIndicator` to the active list.
 - `public void update(float delta):` Loops through all active indicators, calls their internal `update` method, and removes any that are no longer `active`.
 - `public void render(SpriteBatch batch, Camera camera):` Loops through active indicators, projects their 3D `position` to 2D screen coordinates (`camera.project()`), and draws the `text` using the `BitmapFont` and the indicator's `alpha`.

4. `core/src/main/java/com/bpm/minotaur/managers/CombatManager.java` (Modified)

- **Change:** The `attackMonster` method will be modified to return an `AttackResult` object instead of `void` or a simple `boolean`.
- **Logic:** Inside `attackMonster`, after all calculations for hit-chance and damage are complete, it will `return new AttackResult(wasHit, finalDamage);`.

5. `core/src/main/java/com/bpm/minotaur/screens/GameScreen.java` (Modified)

- **Fields:** Add `private DamageIndicatorRenderer damageIndicatorRenderer;`
- **show() Method:** Instantiate the new renderer: `damageIndicatorRenderer = new DamageIndicatorRenderer(hud.getFont());`.
- **handleInput() / Attack Logic:** The section that calls `combatManager.attackMonster(...)` will be updated to:
 1. Receive the `AttackResult`: `AttackResult result = combatManager.attackMonster(...)`
 2. Determine the text string based on the `result`.
 3. Get the monster's position: `targetMonster.position`.
 4. Call the renderer: `damageIndicatorRenderer.showIndicator(text, targetMonster.position);`
- **render(float delta) Method:**
 1. Call `damageIndicatorRenderer.update(delta);` during the update phase.
 2. Add a new 2D rendering pass *after* all 3D rendering is done (after `modelBatch.end()`) but *before* the `hud.render()` call. This pass will use the `game.batch` and the main 3D `camera` to correctly overlay the text.

6. `core/src/main/java/com/bpm/minotaur/rendering/Hud.java` (Modified)

- **Change:** Add a public getter for the `BitmapFont` so `GameScreen` can pass it to the `DamageIndicatorRenderer`.
- **New Method:** `public BitmapFont getFont() { return font; }`

Project Plan & Implementation Steps

To start, I will need the contents of the following files from your repository:

1. `core/src/main/java/com/bpm/minotaur/managers/CombatManager.java`
2. `core/src/main/java/com/bpm/minotaur/screens/GameScreen.java`
3. `core/src/main/java/com/bpm/minotaur/rendering/Hud.java`

Please provide these, and we will begin Step 1.

Step 1: Create Core Data Structures

First, we'll create the two new classes that define the feature's data.

1. Create **AttackResult.java**:

- Create a new file:
`core/src/main/java/com/bpm/minotaur/gamedata/AttackResult.java`
- Add the following code. This class is just a simple container for the combat results.

AttackResult.java

Nov 11, 9:24 AM

2. Create **DamageIndicator.java**:

- Create a new file:
`core/src/main/java/com/bpm/minotaur/rendering/DamageIndicator.java`
- Add the following code. This class manages the state of one floating text element.

DamageIndicator.java

Nov 11, 9:24 AM

Open

Milestone 1: Compile and Run

After you have added these two new files, please **compile and run the game**. The game should run exactly as before with no changes in behavior and no crashes. This confirms the new classes are correctly included in the project.