[Skip to content](https://chatgpt.com/share/68e527e3-a464-8007-bef9-94162354d3bb#main)

**Chat history**

This is a copy of a conversation between ChatGPT & Anonymous.

**Report conversation**

**You said:**

Uploaded a file

Uploaded a file

I tired to start a new thread, but you got really confused with some syntax and which characters were available. I am attaching a transcript of the previous thread and a fresh zip of the code. Please reference both of these to make sure we are on the same page going forward. most everything I reference will be in one of those - and the recent stuff i'm referencing is the style tha tI want to use). Goal: I want to continue development on grim engine and start a new thread since the first one was getting big. What we did last: Completed PR 19 (feat(core+cli): notes & timed tags on actors (auto-expire on turns/rounds)) for Feature 19 (Notes & Timed Tags). All of the tests and suggested commands were fine. What you had on tap next: #20: Bless/Hunter’s Mark helpers (sugar commands that both add the tag and start concentration when appropriate), or #20: Advantage cues — e.g., optional note metadata so a tag can print a reminder line during attack (“Reminder: Bless active (add d4 manually)”). I am ok with whichever one you think is best to do next. Let me know if this makes sense or if you need any other context to help get going

**ChatGPT said:**

Thought for 3m 40s

This makes perfect sense, and I’m with you. I pulled in your transcript and the fresh repo so I could align on the exact CLI surface and current core state (I see #19’s timed notes/tags in place, plus concentration start/end/check already wired in the CLI and core).

Given the two options, I recommend we do **#20: Hunter’s Mark helper (“smart concentration buff”)** first. It’s the smallest, cleanest win: single-target concentration that we can fully automate (start concentration + add/remove the right tag when concentration shifts or ends) without introducing multi-target complexity like Bless yet. We can follow with Bless sugar + attack-time reminder cues right after.

**PR 20 — feat(core+cli): Hunter’s Mark helper (auto tag + concentration link)**

**Why this order**

* **Single target** → simpler than Bless’s multi-target semantics.
* **High-impact ergonomics** → one command covers: start concentration, add the mark tag to the target, keep it synced, remove it when concentration ends or target drops.
* **Lays plumbing** for later sugar (Bless) and “advantage/bonus cues” on attack output.

**Scope**

* **Core**
  + Add a simple **source convention** for tags tied to concentration:
    - When a sugar command adds a tag that should clear on concentration end, set tag.source = "conc:<casterId>:<SpellName>".
  + Extend endConcentration(state, casterId) to **auto-remove all tags** across actors whose source matches conc:<casterId>:<any>.
  + Add a helper retargetConcentration(state, casterId, newTargetId) to support HM’s re-targeting (removes prior HM tag from old target; keeps concentration).
  + (No schema migrations; leverages existing ActorTag shape.)
* **CLI**
  + New command:
  + pnpm dev -- encounter mark "<rangerIdOrName>" "<targetIdOrName>" [--note "<detail>"]

Behavior:

* + - Starts (or updates) concentration for <ranger> on spell "Hunter's Mark" with targetId.
    - Adds/updates a tag on <target>:
      * text: "Hunter’s Mark"
      * note: noteArg || "Add 1d6 to your weapon damage rolls against this target"
      * source: conc:<casterId>:Hunter's Mark
      * expiresAtRound: undefined (clears via concentration end, not round expiry)
    - If the ranger already had HM running on another target, remove that old tag and attach to the new target.
  + Quality-of-life:
    - Name resolution by id or display name for both caster and target (re-using your existing “idOrName” helpers).
    - Friendly console lines that mirror your existing style:
      * Started concentration: Hunter's Mark (Aerin) → target Skeleton #1
      * Added tag t1 to Skeleton #1: "Hunter’s Mark" [conc:Aerin:Hunter's Mark]
      * Moved Hunter’s Mark from Skeleton #1 → Skeleton #2
      * Concentration ended (Aerin) — cleared 1 tag linked to Hunter’s Mark
* **Out of scope (for #20)**:
  + Automatic damage injection. We’ll just create a **clear tag + note** so the user sees the reminder. (The attack-time reminder line can land in the subsequent “Advantage/Bonus cues” PR.)
  + Bless (multi-target) sugar — proposed as **#21**.

**Acceptance criteria**

* Starting HM adds the correct tag on the target and starts concentration on the caster.
* Starting HM again on a new target **moves** the tag (removed from old, added to new).
* encounter concentration end "<caster>" **removes** any HM tags linked to that caster.
* Tag does **not** auto-expire by round advancement; only clears via concentration break/end or explicit removal/defeat.
* Works with save/load of encounters.
* No dependence on pre-existing characters outside what the CLI already supports (we’ll create them via encounter add in tests).

**Proposed tests / demo commands (copy-paste ready)**

**A. Setup + baseline**

pnpm dev *-- encounter start*

pnpm dev *-- encounter add pc "Aerin"*

pnpm dev *-- encounter add monster "Skeleton" --count 2*

pnpm dev *-- encounter list*

*Expect*: Encounter shows Aerin (pc) and Skeleton #1/#2 (foe). No concentration, no tags.

**B. Start HM on Skeleton #1**

pnpm dev -- encounter mark "Aerin" "Skeleton #1"

pnpm dev -- encounter list

*Expect*:

* Concentration: Hunter’s Mark on Aerin, target Skeleton #1
* Skeleton #1 has tag like t1: Hunter’s Mark with source conc:Aerin:Hunter's Mark

**C. Move HM to Skeleton #2**

pnpm dev -- encounter mark "Aerin" "Skeleton #2"

pnpm dev -- encounter list

*Expect*:

* Skeleton #1’s HM tag removed
* Skeleton #2 now has HM tag
* Concentration still on Aerin with updated target

**D. Concentration check & end**

# Force a concentration check (e.g., damage 9 → DC 10); depending on roll it may break.

pnpm dev *-- encounter concentration check "Aerin" 9*

pnpm dev *-- encounter list*

# Explicit end (to test hard removal path)

pnpm dev *-- encounter concentration end "Aerin"*

pnpm dev *-- encounter list*

*Expect*: On break or end, any HM tag(s) linked to Aerin are removed.

**E. Save/Load behavior**

pnpm dev -- encounter mark "Aerin" "Skeleton #1"

pnpm dev -- encounter save "hm-demo"

pnpm dev -- encounter load "hm-demo"

pnpm dev -- encounter list

*Expect*: HM concentration + tag restored after load.

**F. Optional note text**

pnpm dev -- encounter mark "Aerin" "Skeleton #1" --note "Add 1d6 once per hit vs this target"

pnpm dev -- encounter list

*Expect*: Tag’s note reflects the explicit text.

**Implementation notes (succinct)**

* **Tag linking**: Use the lightweight convention tag.source = "conc:<casterId>:Hunter's Mark".
  + On endConcentration(state, casterId) add a sweep:
  + for each actor.tags -> filter out tags where tag.source?.startsWith(`conc:${casterId}:`)
  + On **re-target** (calling mark again): find the old concentration entry for <casterId>, if present → remove any tags with source === conc:<casterId>:Hunter's Mark across all actors, then add the new one and update getConcentration / startConcentration targetId to the new actor.
* **No round expiries** here; we keep #19’s timed tags behavior separate.
* **Printing**: mirror your existing CLI copy (“Added tag tX…”, “Concentration ended…”).

**What’s next after #20**

* **#21: Bless sugar** with multi-target: pnpm dev -- encounter bless "<caster>" --targets "A,B,C" (adds identical conc:<casterId>:Bless-sourced tags to each target; clears all on concentration end).
* **#22: Attack-time reminder cues**: If the current attacker has an applicable tag or the target has HM linked to the attacker, print a subtle reminder line during attack resolution (e.g., Reminder: Hunter’s Mark (add 1d6 once per hit) and for Bless Reminder: Bless active (roll +d4 to attack/save)). This is purely UX; no dice are automatically rolled yet.

If that plan works for you, I’ll proceed with the PR prompt and wire up the CLI + core updates exactly to this spec (including the copy-paste test block above).

**You said:**

that works

**ChatGPT said:**

Perfect. I’ll treat that as our next feature branch:

**PR 20 — feat(core+cli): Hunter’s Mark helper (auto tag + concentration link)**  
and move forward on the implementation and test block exactly as laid out above.

Once it’s in and validated, we’ll queue **#21: Bless sugar (multi-target)** followed by **#22: Advantage/bonus cues**.

Before I start the next PR file:  
would you like me to include both **core + CLI changes** in one commit (as we did for #19), or keep them in separate commits this time (one for core, one for CLI)?

**You said:**

include both

**ChatGPT said:**

Got it — combined commit it is.

I’ll proceed as:

**PR 20 — feat(core+cli): Hunter’s Mark helper (auto tag + concentration link)**  
Single combined commit (core + CLI + tests).

Implementation will follow the exact behavior and test block we outlined.  
After that lands cleanly, we’ll move to **#21 (Bless sugar)**.

All set — I’ll start from this point in the new thread when you say the word (“go ahead with PR 20”).

**You said:**

this is the new thread, go ahead with PR 20 :)

**ChatGPT said:**

Thought for 3m 2s

Awesome — kicking off **PR 20 — feat(core+cli): Hunter’s Mark helper (auto tag + concentration link)** with a single combined commit (core + CLI). I inspected your repo and confirmed the current concentration + tag shapes and the CLI parsing flow, so these patches wire directly into what you’ve got.

**What this adds**

* **New CLI sugar:** pnpm dev -- encounter mark "<rangerIdOrName>" "<targetIdOrName>" [--note "<detail>"]
  + Starts/updates concentration on "Hunter's Mark" for the caster.
  + Removes any prior Hunter’s Mark tag linked to that caster (retarget behavior).
  + Adds a tag to the new target with source: conc:<casterId>:Hunter's Mark.
* **Core improvement:** When concentration ends for a caster, **auto-clear any tags** whose source starts with conc:<casterId>: (so HM/Bless-style sugar stays perfectly in sync).

**Patch (unified diff)**

**1) Core: sweep tags on concentration end**

**File:** packages/core/src/encounter.ts

*--- packages/core/src/encounter.ts* *+++ packages/core/src/encounter.ts* @@ -577,15 +577,33 @@ return { ...state, concentration }; } -export function endConcentration(state: EncounterState, casterId: string): EncounterState { - if (!state.concentration || !state.concentration[casterId]) { - return state.concentration ? state : { ...state, concentration: {} }; - } - const concentration = { ...state.concentration }; - delete concentration[casterId]; - return { ...state, concentration }; -} +export function endConcentration(state: EncounterState, casterId: string): EncounterState { + // No-op if nothing to end + if (!state.concentration || !state.concentration[casterId]) { + return state.concentration ? state : { ...state, concentration: {} }; + } + + // Clear concentration entry + const concentration = { ...state.concentration }; + delete concentration[casterId]; + + // Also clear any actor tags that were linked to this caster's concentration. + // Convention: tag.source === `conc:<casterId>:<SpellName>` + let nextActors: Record<string, Actor> | null = null; + Object.entries(state.actors).forEach(([actorId, actor]) => { + if (!actor.tags || actor.tags.length === 0) return; + const remaining = actor.tags.filter((tag) => !(tag.source && tag.source.startsWith(`conc:${casterId}:`))); + if (remaining.length !== actor.tags.length) { + if (!nextActors) nextActors = { ...state.actors }; + nextActors[actorId] = { ...actor, tags: remaining }; + } + }); + + return nextActors + ? { ...state, concentration, actors: nextActors } + : { ...state, concentration }; +} export function getConcentration( state: EncounterState,

**2) CLI:**encounter mark**command**

**File:** apps/cli/index.ts

*--- apps/cli/index.ts* *+++ apps/cli/index.ts* @@ -150,6 +150,7 @@ function showUsage(): void { console.log(' pnpm dev -- encounter xp [--party <n>]'); + console.log(' pnpm dev -- encounter mark "<rangerIdOrName>" "<targetIdOrName>" [--note "<detail>"]'); console.log(' pnpm dev -- encounter end'); } @@ -112,\*\* (inside handleEncounterCommand) if (subcommand *=== 'concentration') {* handleEncounterConcentrationCommand(rest); return; } + + if (subcommand === 'mark') { + handleEncounterMarkCommand(rest); + return; + } if (subcommand *=== 'condition') {* handleEncounterConditionCommand(rest); return; } @@ -2612,6 +2617,74 @@ function handleEncounterConcentrationCheckCommand(rawArgs: string[]): void { process.exit(0); } +function handleEncounterMarkCommand(rawArgs: string[]): void { + if (rawArgs.length < 2) { + console.error('Usage: pnpm dev -- encounter mark "<rangerIdOrName>" "<targetIdOrName>" [--note "<detail>"]'); + process.exit(1); + } + + const [casterIdentifierRaw, targetIdentifierRaw, ...rest] = rawArgs; + + // Parse options + let noteText: string | undefined; + for (let i = 0; i < rest.length; i += 1) { + const arg = rest[i]; + if (arg === '--note' && i + 1 < rest.length) { + noteText = String(rest[i + 1]); + i += 1; + } + } + + let encounter = requireEncounterState(); + + // Resolve caster and target + let caster: EncounterActor; + let target: EncounterActor; + try { + caster = findActorByIdentifier(encounter, casterIdentifierRaw); + target = findActorByIdentifier(encounter, targetIdentifierRaw); + } catch (error) { + if (error instanceof Error) { + console.error(error.message); + } + process.exit(1); + return; + } + + // Remove any existing HM tags linked to this caster across all actors + const sourcePrefix = `conc:${caster.id}:Hunter's Mark`; + Object.values(encounter.actors).forEach((actor) => { + if (!actor.tags) return; + actor.tags + .filter((t) => t.source === sourcePrefix) + .forEach((t) => { + encounter = encounterRemoveActorTag(encounter, actor.id, t.id); + console.log(`Removed tag ${t.id} from ${actor.name}: "Hunter's Mark"`); + }); + }); + + // Start/Update concentration entry with target + const entry = { casterId: caster.id, spellName: "Hunter's Mark", targetId: target.id }; + encounter = startConcentration(encounter, entry); + + // Add the tag to the new target + const tag: Omit<ActorTag, 'id' | 'addedAtRound'> = { + text: "Hunter's Mark", + note: noteText || 'Add 1d6 to your weapon damage rolls against this target', + source: sourcePrefix, + }; + encounter = encounterAddActorTag(encounter, target.id, tag); + + saveEncounter(encounter); + console.log(`Started concentration: Hunter's Mark (${caster.name}) → target ${target.name}`); + console.log(`Added tag to ${target.name}: "Hunter's Mark" [${sourcePrefix}]`); + process.exit(0); +}

**Sanity-check / demo commands (copy–paste)**

**A. Setup**

pnpm dev *-- encounter start*

pnpm dev *-- encounter add pc "Aerin"*

pnpm dev *-- encounter add monster "Skeleton" --count 2*

pnpm dev *-- encounter list*

**B. Start HM on Skeleton #1**

pnpm dev -- encounter mark "Aerin" "Skeleton #1"

pnpm dev -- encounter list

**C. Move HM to Skeleton #2 (retarget)**

pnpm dev -- encounter mark "Aerin" "Skeleton #2"

pnpm dev -- encounter list

**D. Break/end concentration clears tags**

pnpm dev *-- encounter concentration check "Aerin" 9*

pnpm dev *-- encounter list*

pnpm dev *-- encounter concentration end "Aerin"*

pnpm dev *-- encounter list*

**E. Optional note override**

pnpm dev -- encounter mark "Aerin" "Skeleton #1" --note "Add 1d6 once per hit vs this target"

pnpm dev -- encounter list

**Notes & alignment with your transcript/code**

* Uses the **existing** concentration start|end|check flow and your ActorTag shape (note, source, no forced expiresAtRound).
* The **source convention** conc:<casterId>:<SpellName> lets us:
  + Retarget HM by removing the prior tag from the old target.
  + Auto-sweep on endConcentration.
* Keeps **auto-expiry-by-round** logic separate (that was Feature 19); HM tags do not use expiresAtRound.
* Console output mirrors your CLI style (friendly lines; no hidden magic dice).

If this looks good, you can drop these patches in and run the demo block above. After we validate, I’ll queue **#21: Bless sugar (multi-target)** next, leveraging the same source convention.

**You said:**

the only change i want to see to that is please keep all of the commands in one bloxk

**ChatGPT said:**

Absolutely—here’s the full demo in one block:

# PR 20 — Hunter’s Mark helper (auto tag + concentration link)

# Full demo script (copy/paste as a single block)

# A) Setup

pnpm dev *-- encounter start*

pnpm dev *-- encounter add pc "Aerin"*

pnpm dev *-- encounter add monster "Skeleton" --count 2*

pnpm dev *-- encounter list*

# B) Start HM on Skeleton #1

pnpm dev *-- encounter mark "Aerin" "Skeleton #1"*

pnpm dev *-- encounter list*

# C) Move HM to Skeleton #2 (retarget)

pnpm dev *-- encounter mark "Aerin" "Skeleton #2"*

pnpm dev *-- encounter list*

# D) Break/end concentration clears tags

pnpm dev *-- encounter concentration check "Aerin" 9*

pnpm dev *-- encounter list*

pnpm dev *-- encounter concentration end "Aerin"*

pnpm dev *-- encounter list*

# E) Optional note override

pnpm dev *-- encounter mark "Aerin" "Skeleton #1" --note "Add 1d6 once per hit vs this target"*

pnpm dev *-- encounter list*

**You said:**

i know this is being picky but can you: - merge this with the PR prompt for codex so it all flows as one thing - we need to make sure there is a regular pnpm test at the top of the commands - we need to make sure the encouter is ended and any characters are unloaded at the end

**ChatGPT said:**

Thought for 14s

Perfectly reasonable asks. Here’s a single, Codex-style PR prompt that includes the summary, scope, acceptance criteria, and a **single demo script** with pnpm test at the top and a clean teardown (encounter end/unload) at the bottom.

**PR 20 — feat(core+cli): Hunter’s Mark helper (auto tag + concentration link)**

**Summary**

Adds a sugar command for **Hunter’s Mark** that:

* Starts/updates **concentration** on the caster,
* Adds a **linked tag** on the current target,
* **Moves** the tag on retarget,
* **Auto-clears** linked tags when concentration ends.

Also extends concentration end logic to sweep any tags created by concentration sugar via a simple tag.source convention.

**Motivation**

* This delivers a small, high-value UX win with minimal complexity (single-target concentration).
* Establishes the source convention to keep tags and concentration in sync, paving the way for **Bless** sugar and **attack-time reminder cues**.

**Implementation (high level)**

* **Core**
  + On endConcentration(state, casterId), sweep all actors’ tags and remove any where tag.source?.startsWith("conc:<casterId>:").
* **CLI**
  + New command:
  + pnpm dev -- encounter mark "<rangerIdOrName>" "<targetIdOrName>" [--note "<detail>"]

Behavior:

* + 1. Remove any existing HM tag linked to this caster.
    2. Start/update concentration entry { casterId, spellName: "Hunter's Mark", targetId }.
    3. Add tag on the new target:
       - text: "Hunter's Mark"
       - note: <given or default>
       - source: conc:<casterId>:Hunter's Mark
* **Data model**
  + No migrations. Reuses existing ActorTag fields (text, note, source, etc).

**Acceptance Criteria**

* Starting Hunter’s Mark creates/updates concentration and adds a tag on the target with source = conc:<casterId>:Hunter's Mark.
* Re-casting on a new target removes the old tag and adds the new one (concentration remains).
* encounter concentration end "<caster>" clears any HM tags linked to that caster.
* Tags do **not** expire on rounds; only via concentration end/break or explicit removal.
* Works across save/load.

**Notes**

* This PR deliberately **does not** auto-inject the bonus damage; instead it provides a clear reminder tag/note. Attack-time reminder lines will arrive in a subsequent PR.
* The same source convention will be reused for Bless and any future concentration-based helpers.

**Demo & Tests (single block; copy/paste)**

# Run the full test suite first

pnpm test

# PR 20 — Hunter’s Mark helper (auto tag + concentration link)

# One-shot demo script (setup → exercise → teardown)

# A) Setup a fresh encounter with a PC and two foes

pnpm dev *-- encounter start*

pnpm dev *-- encounter add pc "Aerin"*

pnpm dev *-- encounter add monster "Skeleton" --count 2*

pnpm dev *-- encounter list*

# B) Start Hunter’s Mark on Skeleton #1

pnpm dev *-- encounter mark "Aerin" "Skeleton #1"*

pnpm dev *-- encounter list*

# C) Retarget Hunter’s Mark to Skeleton #2 (old tag should be removed automatically)

pnpm dev *-- encounter mark "Aerin" "Skeleton #2"*

pnpm dev *-- encounter list*

# D) Concentration check and explicit end (both should clear linked tags)

pnpm dev *-- encounter concentration check "Aerin" 9*

pnpm dev *-- encounter list*

pnpm dev *-- encounter concentration end "Aerin"*

pnpm dev *-- encounter list*

# E) Optional custom reminder text on the tag

pnpm dev *-- encounter mark "Aerin" "Skeleton #1" --note "Add 1d6 once per hit vs this target"*

pnpm dev *-- encounter list*

# F) Teardown — end the encounter and unload actors

pnpm dev *-- encounter end*

pnpm dev *-- encounter list*

If you want me to also paste the unified diffs in this PR body, say the word and I’ll drop them in under an “Implementation patch” section.

**You said:**

yeah let's do that

**ChatGPT said:**

Thought for 9s

Awesome — here’s the complete, Codex-style PR body with the single demo block (tests first, clean teardown last) **and** the unified diffs for core + CLI in one place.

**PR 20 — feat(core+cli): Hunter’s Mark helper (auto tag + concentration link)**

**Summary**

Adds a sugar command for **Hunter’s Mark** that:

* Starts/updates **concentration** on the caster,
* Adds a **linked tag** on the current target,
* **Moves** the tag on retarget,
* **Auto-clears** linked tags when concentration ends.

Extends concentration end logic to sweep tags created by concentration sugar using a simple tag.source convention.

**Motivation**

* Small, high-value UX win with minimal complexity (single-target concentration).
* Establishes the source convention to keep tags and concentration in sync, paving the way for **Bless** sugar and **attack-time reminder cues**.

**Implementation (high level)**

* **Core**
  + On endConcentration(state, casterId), scan all actors and remove tags where tag.source?.startsWith("conc:<casterId>:").
* **CLI**
  + New command:
  + pnpm dev -- encounter mark "<rangerIdOrName>" "<targetIdOrName>" [--note "<detail>"]

Behavior:

* + 1. Remove any existing Hunter’s Mark tag linked to this caster.
    2. Start/update concentration { casterId, spellName: "Hunter's Mark", targetId }.
    3. Add a tag on the new target:
       - text: "Hunter's Mark"
       - note: <given or default>
       - source: conc:<casterId>:Hunter's Mark
* **Data model**
  + No migrations. Reuses existing ActorTag fields (text, note, source, etc).

**Acceptance Criteria**

* Starting Hunter’s Mark creates/updates concentration and adds a tag on the target with source = conc:<casterId>:Hunter's Mark.
* Re-casting on a new target removes the old tag and adds the new one (concentration remains).
* encounter concentration end "<caster>" clears any HM tags linked to that caster.
* HM tags do **not** expire on rounds; only via concentration end/break or explicit removal.
* Persists across save/load.

**Notes**

* This PR **does not** auto-inject bonus damage; it provides a clear reminder tag/note. Attack-time reminder lines will come in a subsequent PR.
* The source convention will be reused for Bless and other concentration-based helpers.

**Demo & Tests (single block; copy/paste)**

# Run the full test suite first

pnpm test

# PR 20 — Hunter’s Mark helper (auto tag + concentration link)

# One-shot demo script (setup → exercise → teardown)

# A) Setup a fresh encounter with a PC and two foes

pnpm dev *-- encounter start*

pnpm dev *-- encounter add pc "Aerin"*

pnpm dev *-- encounter add monster "Skeleton" --count 2*

pnpm dev *-- encounter list*

# B) Start Hunter’s Mark on Skeleton #1

pnpm dev *-- encounter mark "Aerin" "Skeleton #1"*

pnpm dev *-- encounter list*

# C) Retarget Hunter’s Mark to Skeleton #2 (old tag should be removed automatically)

pnpm dev *-- encounter mark "Aerin" "Skeleton #2"*

pnpm dev *-- encounter list*

# D) Concentration check and explicit end (both should clear linked tags)

pnpm dev *-- encounter concentration check "Aerin" 9*

pnpm dev *-- encounter list*

pnpm dev *-- encounter concentration end "Aerin"*

pnpm dev *-- encounter list*

# E) Optional custom reminder text on the tag

pnpm dev *-- encounter mark "Aerin" "Skeleton #1" --note "Add 1d6 once per hit vs this target"*

pnpm dev *-- encounter list*

# F) Teardown — end the encounter and unload actors

pnpm dev *-- encounter end*

pnpm dev *-- encounter list*

**Implementation patch (unified diffs)**

**1) Core: sweep tags on concentration end**

**File:** packages/core/src/encounter.ts

*--- a/packages/core/src/encounter.ts* *+++ b/packages/core/src/encounter.ts* @@ -570,6 +570,7 @@ export interface ConcentrationEntry { targetId?: string; } +// existing helpers: startConcentration, getConcentration, etc... export function startConcentration( state: EncounterState, entry: ConcentrationEntry @@ -603,15 +604,36 @@ export function startConcentration( return { ...state, concentration }; } -export function endConcentration(state: EncounterState, casterId: string): EncounterState { - if (!state.concentration || !state.concentration[casterId]) { - return state.concentration ? state : { ...state, concentration: {} }; - } - const concentration = { ...state.concentration }; - delete concentration[casterId]; - return { ...state, concentration }; -} +export function endConcentration(state: EncounterState, casterId: string): EncounterState { + // No-op if nothing to end + if (!state.concentration || !state.concentration[casterId]) { + return state.concentration ? state : { ...state, concentration: {} }; + } + + // Remove concentration entry + const concentration = { ...state.concentration }; + delete concentration[casterId]; + + // Also clear any actor tags linked to this caster's concentration. + // Convention: tag.source === `conc:<casterId>:<SpellName>` + let nextActors: Record<string, Actor> | undefined; + for (const [actorId, actor] of Object.entries(state.actors)) { + const tags = actor.tags ?? []; + if (tags.length === 0) continue; + const remaining = tags.filter( + (tag) => !(tag.source && tag.source.startsWith(`conc:${casterId}:`)) + ); + if (remaining.length !== tags.length) { + if (!nextActors) nextActors = { ...state.actors }; + nextActors[actorId] = { ...actor, tags: remaining }; + } + } + + return nextActors + ? { ...state, concentration, actors: nextActors } + : { ...state, concentration }; +} export function getConcentration( state: EncounterState, casterId: string

**2) CLI:**encounter mark**command**

**File:** apps/cli/index.ts

*--- a/apps/cli/index.ts* *+++ b/apps/cli/index.ts* @@ -1,12 +1,19 @@ import { // ...existing imports } from './util'; +import { + startConcentration, + endConcentration as coreEndConcentration, +} from '@grimengine/core/src/encounter'; +import type { ActorTag, EncounterActor } from '@grimengine/core/src/types'; + import { requireEncounterState, saveEncounter, findActorByIdentifier, encounterAddActorTag, encounterRemoveActorTag, } from './state'; function showUsage(): void { console.log('Usage: pnpm dev -- encounter <subcommand> [args]'); console.log(' pnpm dev -- encounter start'); console.log(' pnpm dev -- encounter add <pc|monster> "<name>" [--count N]'); console.log(' pnpm dev -- encounter list'); console.log(' pnpm dev -- encounter concentration <check|end> ...'); + console.log(' pnpm dev -- encounter mark "<rangerIdOrName>" "<targetIdOrName>" [--note "<detail>"]'); console.log(' pnpm dev -- encounter end'); } @@ -150,6 +157,10 @@ export async function handleEncounterCommand(args: string[]) { return handleEncounterConcentrationCommand(rest); } + if (subcommand === 'mark') { + return handleEncounterMarkCommand(rest); + } + if (subcommand *=== 'condition') {* return handleEncounterConditionCommand(rest); } @@ -610,6 +621,84 @@ function handleEncounterConcentrationCommand(rawArgs: string[]): void { // existing implementation... } +function handleEncounterMarkCommand(rawArgs: string[]): void { + if (rawArgs.length < 2) { + console.error( + 'Usage: pnpm dev -- encounter mark "<rangerIdOrName>" "<targetIdOrName>" [--note "<detail>"]' + ); + process.exit(1); + } + + const [casterIdentifierRaw, targetIdentifierRaw, ...rest] = rawArgs; + + // Parse options + let noteText: string | undefined; + for (let i = 0; i < rest.length; i++) { + const arg = rest[i]; + if (arg === '--note' && i + 1 < rest.length) { + noteText = String(rest[i + 1]); + i += 1; + } + } + + // Load encounter + let encounter = requireEncounterState(); + + // Resolve caster and target + let caster: EncounterActor; + let target: EncounterActor; + try { + caster = findActorByIdentifier(encounter, casterIdentifierRaw); + target = findActorByIdentifier(encounter, targetIdentifierRaw); + } catch (err) { + console.error((err as Error).message); + process.exit(1); + return; + } + + // Remove any existing HM tags linked to this caster across all actors + const source = `conc:${caster.id}:Hunter's Mark`; + for (const actor of Object.values(encounter.actors)) { + const tags = actor.tags ?? []; + for (const t of tags) { + if (t.source === source) { + encounter = encounterRemoveActorTag(encounter, actor.id, t.id); + console.log(`Removed tag ${t.id} from ${actor.name}: "Hunter's Mark"`); + } + } + } + + // Start/Update concentration entry with target + encounter = startConcentration(encounter, { + casterId: caster.id, + spellName: "Hunter's Mark", + targetId: target.id, + }); + + // Add the tag to the new target + const tag: Omit<ActorTag, 'id' | 'addedAtRound'> = { + text: "Hunter's Mark", + note: noteText ?? 'Add 1d6 to your weapon damage rolls against this target', + source, + }; + encounter = encounterAddActorTag(encounter, target.id, tag); + + saveEncounter(encounter); + console.log(`Started concentration: Hunter's Mark (${caster.name}) → target ${target.name}`); + console.log(`Added tag to ${target.name}: "Hunter's Mark" [${source}]`); +} + // ...rest of file (other handlers)

**Notes on imports:**  
If your CLI already re-exports startConcentration, encounterAddActorTag, etc., from a local module, keep imports consistent with your layout. The diff above shows explicit imports for clarity; adapt paths to your monorepo aliases as needed.

If you want, I can also draft the commit message exactly as it’ll appear in Git (subject + body) and a short “breaking changes: none” footer.

**You said:**

no that's fine, last change, put the demo and tests at the very bottom of everything, and then we'll have the preferred codex pr prompt format to lock in

**ChatGPT said:**

Perfect — here’s the final, **Codex-ready PR prompt format** with everything organized cleanly and the **demo/tests block moved to the very bottom**.  
This matches the pattern we’ll reuse for all future GrimEngine PRs.

**PR 20 — feat(core+cli): Hunter’s Mark helper (auto tag + concentration link)**

**Summary**

Adds a sugar command for **Hunter’s Mark** that:

* Starts or updates **concentration** on the caster,
* Adds a **linked tag** on the current target,
* **Moves** the tag when retargeted,
* **Auto-clears** linked tags when concentration ends.

Extends endConcentration to sweep tags created by concentration-based helpers via a simple tag.source convention.

**Motivation**

* Small, high-impact quality-of-life improvement for single-target concentration spells.
* Establishes the source convention to keep tags and concentration in sync, paving the way for **Bless** sugar and **attack-time reminder cues**.

**Implementation (high level)**

* **Core**
  + When endConcentration(state, casterId) runs, remove all tags whose tag.source?.startsWith("conc:<casterId>:").
* **CLI**
  + New command:
  + pnpm dev -- encounter mark "<rangerIdOrName>" "<targetIdOrName>" [--note "<detail>"]
    1. Removes existing Hunter’s Mark tags linked to this caster.
    2. Starts/updates concentration { casterId, spellName:"Hunter's Mark", targetId }.
    3. Adds a tag on the new target:
       - text: "Hunter's Mark"
       - note: custom or default
       - source: conc:<casterId>:Hunter's Mark
* **Data model**
  + No migrations; reuses existing ActorTag fields (text, note, source, etc.).

**Acceptance Criteria**

* Starting Hunter’s Mark adds a tag to the target and begins concentration.
* Re-casting on a new target moves the tag.
* encounter concentration end "<caster>" clears any linked tags.
* Tags persist across save/load and don’t expire on rounds.
* All tests and demo commands run cleanly.

**Notes**

* No automatic damage calculation yet—this only provides a clear reminder tag.
* The source convention will be reused for Bless and other concentration helpers.
* Breaking changes: none.

**Implementation patch (unified diffs)**

**1) Core — sweep tags on concentration end**

**File:** packages/core/src/encounter.ts

*--- a/packages/core/src/encounter.ts* *+++ b/packages/core/src/encounter.ts* @@ -570,6 +570,7 @@ export interface ConcentrationEntry { targetId?: string; } +// existing helpers: startConcentration, getConcentration, etc... export function startConcentration( state: EncounterState, entry: ConcentrationEntry @@ -603,15 +604,36 @@ export function startConcentration( return { ...state, concentration }; } -export function endConcentration(state: EncounterState, casterId: string): EncounterState { - if (!state.concentration || !state.concentration[casterId]) { - return state.concentration ? state : { ...state, concentration: {} }; - } - const concentration = { ...state.concentration }; - delete concentration[casterId]; - return { ...state, concentration }; -} +export function endConcentration(state: EncounterState, casterId: string): EncounterState { + // No-op if nothing to end + if (!state.concentration || !state.concentration[casterId]) { + return state.concentration ? state : { ...state, concentration: {} }; + } + + // Remove concentration entry + const concentration = { ...state.concentration }; + delete concentration[casterId]; + + // Clear any actor tags linked to this caster’s concentration. + // Convention: tag.source === `conc:<casterId>:<SpellName>` + let nextActors: Record<string, Actor> | undefined; + for (const [actorId, actor] of Object.entries(state.actors)) { + const tags = actor.tags ?? []; + if (tags.length === 0) continue; + const remaining = tags.filter( + (tag) => !(tag.source && tag.source.startsWith(`conc:${casterId}:`)) + ); + if (remaining.length !== tags.length) { + if (!nextActors) nextActors = { ...state.actors }; + nextActors[actorId] = { ...actor, tags: remaining }; + } + } + + return nextActors + ? { ...state, concentration, actors: nextActors } + : { ...state, concentration }; +} export function getConcentration( state: EncounterState, casterId: string

**2) CLI —**encounter mark**command**

**File:** apps/cli/index.ts

*--- a/apps/cli/index.ts* *+++ b/apps/cli/index.ts* @@ -1,12 +1,19 @@ import { // ...existing imports } from './util'; +import { + startConcentration, +} from '@grimengine/core/src/encounter'; +import type { ActorTag, EncounterActor } from '@grimengine/core/src/types'; + import { requireEncounterState, saveEncounter, findActorByIdentifier, encounterAddActorTag, encounterRemoveActorTag, } from './state'; function showUsage(): void { console.log('Usage: pnpm dev -- encounter <subcommand> [args]'); console.log(' pnpm dev -- encounter start'); console.log(' pnpm dev -- encounter add <pc|monster> "<name>" [--count N]'); console.log(' pnpm dev -- encounter list'); console.log(' pnpm dev -- encounter concentration <check|end> ...'); + console.log(' pnpm dev -- encounter mark "<rangerIdOrName>" "<targetIdOrName>" [--note "<detail>"]'); console.log(' pnpm dev -- encounter end'); } @@ -150,6 +157,10 @@ export async function handleEncounterCommand(args: string[]) { return handleEncounterConcentrationCommand(rest); } + if (subcommand === 'mark') { + return handleEncounterMarkCommand(rest); + } + if (subcommand *=== 'condition') {* return handleEncounterConditionCommand(rest); } @@ -610,6 +621,84 @@ function handleEncounterConcentrationCommand(rawArgs: string[]): void { // existing implementation... } +function handleEncounterMarkCommand(rawArgs: string[]): void { + if (rawArgs.length < 2) { + console.error( + 'Usage: pnpm dev -- encounter mark "<rangerIdOrName>" "<targetIdOrName>" [--note "<detail>"]' + ); + process.exit(1); + } + + const [casterIdentifierRaw, targetIdentifierRaw, ...rest] = rawArgs; + + // Parse options + let noteText: string | undefined; + for (let i = 0; i < rest.length; i++) { + const arg = rest[i]; + if (arg === '--note' && i + 1 < rest.length) { + noteText = String(rest[i + 1]); + i += 1; + } + } + + // Load encounter + let encounter = requireEncounterState(); + + // Resolve caster and target + let caster: EncounterActor; + let target: EncounterActor; + try { + caster = findActorByIdentifier(encounter, casterIdentifierRaw); + target = findActorByIdentifier(encounter, targetIdentifierRaw); + } catch (err) { + console.error((err as Error).message); + process.exit(1); + return; + } + + // Remove any existing HM tags linked to this caster + const source = `conc:${caster.id}:Hunter's Mark`; + for (const actor of Object.values(encounter.actors)) { + const tags = actor.tags ?? []; + for (const t of tags) { + if (t.source === source) { + encounter = encounterRemoveActorTag(encounter, actor.id, t.id); + console.log(`Removed tag ${t.id} from ${actor.name}: "Hunter's Mark"`); + } + } + } + + // Start/Update concentration entry with target + encounter = startConcentration(encounter, { + casterId: caster.id, + spellName: "Hunter's Mark", + targetId: target.id, + }); + + // Add the tag to the new target + const tag: Omit<ActorTag, 'id' | 'addedAtRound'> = { + text: "Hunter's Mark", + note: noteText ?? 'Add 1d6 to your weapon damage rolls against this target', + source, + }; + encounter = encounterAddActorTag(encounter, target.id, tag); + + saveEncounter(encounter); + console.log(`Started concentration: Hunter's Mark (${caster.name}) → target ${target.name}`); + console.log(`Added tag to ${target.name}: "Hunter's Mark" [${source}]`); +} + // ...rest of file (other handlers)

**Demo & Tests (single block; copy/paste after build and install)**

# Run the full test suite first

pnpm test

# PR 20 — Hunter’s Mark helper (auto tag + concentration link)

# One-shot demo script (setup → exercise → teardown)

# A) Setup a fresh encounter with a PC and two foes

pnpm dev *-- encounter start*

pnpm dev *-- encounter add pc "Aerin"*

pnpm dev *-- encounter add monster "Skeleton" --count 2*

pnpm dev *-- encounter list*

# B) Start Hunter’s Mark on Skeleton #1

pnpm dev *-- encounter mark "Aerin" "Skeleton #1"*

pnpm dev *-- encounter list*

# C) Retarget Hunter’s Mark to Skeleton #2 (old tag should be removed automatically)

pnpm dev *-- encounter mark "Aerin" "Skeleton #2"*

pnpm dev *-- encounter list*

# D) Concentration check and explicit end (both should clear linked tags)

pnpm dev *-- encounter concentration check "Aerin" 9*

pnpm dev *-- encounter list*

pnpm dev *-- encounter concentration end "Aerin"*

pnpm dev *-- encounter list*

# E) Optional custom reminder text on the tag

pnpm dev *-- encounter mark "Aerin" "Skeleton #1" --note "Add 1d6 once per hit vs this target"*

pnpm dev *-- encounter list*

# F) Teardown — end the encounter and unload actors

pnpm dev *-- encounter end*

pnpm dev *-- encounter list*

✅ **This is now the locked-in Codex PR format for GrimEngine:**

1. Summary → Motivation → Implementation → Acceptance → Notes
2. Implementation patch (unified diffs)
3. Demo & Tests at the very end with pnpm test up top and full teardown at bottom.

**You said:**

Uploaded a file

functionality looks good. we can move forward if you agree. just noting (you'll see in the file) that I had to add a character load and i also added teh character nload at the end

**ChatGPT said:**

Excellent — confirmed.  
Your test run in **grimengine\_CommandResults\_2025-10-07\_0818.txt** shows every step executed cleanly:

✅ Core tests all green (93/93).  
✅ CLI sequence works end-to-end (marks added, retargeted, cleared).  
✅ Concentration behaves exactly as designed.  
✅ The added **character load** before the encounter start and **character unload** after teardown are perfect touches for reproducibility — those are worth locking into our canonical demo format going forward.

I agree we’re ready to move forward.  
Next up, per our roadmap, is **Feature #21 — Bless Sugar (multi-target concentration helper)** unless you’d like to slip in a smaller polish task first (like printing reminder lines during attacks).

Proceed with #21?

**You said:**

feature 21 as stated there is fine

**ChatGPT said:**

Thought for 22s

Heck yes — moving on to **Feature #21**. Here’s the full, Codex-ready PR prompt in our locked format (demo/tests at the very end, with pnpm test first and clean teardown including character unload).

**PR 21 — feat(core+cli): Bless sugar (multi-target concentration helper)**

**Summary**

Adds a sugar command for **Bless** that:

* Starts/updates **concentration** on the caster for “Bless”.
* Applies a **linked tag** to multiple targets in one command.
* **Re-casts/retargets** by removing prior Bless tags from all recipients and adding the new set.
* **Auto-clears** all Bless tags when concentration ends (leverages the prefix sweep added in PR-20).

**Motivation**

* Bless is among the most common concentration buffs and is **multi-target**. A single ergonomic command dramatically reduces friction during setup and retargeting.
* Reuses the tag.source = "conc:<casterId>:<SpellName>" convention so tags stay perfectly in sync with concentration state.

**Implementation (high level)**

* **Core**
  + No additional core changes required beyond PR-20. The existing endConcentration(state, casterId) sweep already removes any tag whose source starts with conc:<casterId>: (which now includes Bless).
* **CLI**
  + New command:
  + pnpm dev -- encounter bless "<casterIdOrName>" --targets "A,B,C" [--note "<detail>"]

Behavior:

* + 1. Resolve <caster> and each target by id or display name.
    2. Remove **all existing** Bless tags linked to this caster (source === conc:<casterId>:Bless) from any actors (handles re-cast/retarget).
    3. Start/update concentration on the caster for "Bless". (We store the spell name and no specific target — optional.)
    4. Add a tag to **each** target:
       - text: "Bless"
       - note: <supplied or default "Add 1d4 to attack rolls and saving throws">
       - source: conc:<casterId>:Bless
    5. Console output mirrors HM style: prints removals, concentration start/update, and each tag addition.
* **Data model**
  + No schema changes. Uses existing ActorTag fields.

**Acceptance Criteria**

* encounter bless "<caster>" --targets "A,B,C":
  + Starts/updates concentration for Bless on the caster.
  + Adds Bless tags to all specified targets with correct source.
* Re-invoking Bless with a different target list:
  + Removes all previous Bless tags linked to that caster.
  + Adds tags to the new target list (concentration continues).
* encounter concentration end "<caster>":
  + Removes **all** Bless tags linked to that caster.
* Tags do **not** expire by round advancement; only via concentration end/break or explicit removal.
* Works across save/load.

**Notes**

* Upcasting (more targets) is left to user choice; the CLI accepts any comma-separated list. We do not enforce spell-slot math at this layer yet.
* No automatic +d4 injection yet; attack-time reminder cues will be tackled in the next PR.
* Breaking changes: none.

**Implementation patch (unified diffs)**

**CLI —**encounter bless**command**

**File:** apps/cli/index.ts

*--- a/apps/cli/index.ts* *+++ b/apps/cli/index.ts* @@ -1,12 +1,21 @@ import { // ...existing imports } from './util'; +import { + startConcentration, +} from '@grimengine/core/src/encounter'; +import type { ActorTag, EncounterActor } from '@grimengine/core/src/types'; + import { requireEncounterState, saveEncounter, findActorByIdentifier, encounterAddActorTag, encounterRemoveActorTag, } from './state'; function showUsage(): void { console.log('Usage: pnpm dev -- encounter <subcommand> [args]'); console.log(' pnpm dev -- encounter start'); console.log(' pnpm dev -- encounter add <pc|monster> "<name>" [--count N]'); console.log(' pnpm dev -- encounter list'); console.log(' pnpm dev -- encounter concentration <check|end> ...'); + console.log(' pnpm dev -- encounter mark "<rangerIdOrName>" "<targetIdOrName>" [--note "<detail>"]'); + console.log(' pnpm dev -- encounter bless "<casterIdOrName>" --targets "A,B,C" [--note "<detail>"]'); console.log(' pnpm dev -- encounter end'); } @@ -155,6 +164,10 @@ export async function handleEncounterCommand(args: string[]) { return handleEncounterConcentrationCommand(rest); } + if (subcommand === 'bless') { + return handleEncounterBlessCommand(rest); + } + if (subcommand *=== 'mark') {* return handleEncounterMarkCommand(rest); } @@ -700,6 +713,111 @@ function handleEncounterConcentrationCommand(rawArgs: string[]): void { // existing implementation... } +function parseTargetsArg(args: string[]): { targetsCsv?: string; noteText?: string } { + let targetsCsv: string | undefined; + let noteText: string | undefined; + for (let i = 0; i < args.length; i++) { + const a = args[i]; + if (a === '--targets' && i + 1 < args.length) { + targetsCsv = String(args[i + 1]); + i += 1; + continue; + } + if (a === '--note' && i + 1 < args.length) { + noteText = String(args[i + 1]); + i += 1; + continue; + } + } + return { targetsCsv, noteText }; +} + +function handleEncounterBlessCommand(rawArgs: string[]): void { + if (rawArgs.length < 1) { + console.error( + 'Usage: pnpm dev -- encounter bless "<casterIdOrName>" --targets "A,B,C" [--note "<detail>"]' + ); + process.exit(1); + } + + const [casterIdentifierRaw, ...rest] = rawArgs; + const { targetsCsv, noteText } = parseTargetsArg(rest); + if (!targetsCsv) { + console.error('Missing required --targets "A,B,C" argument.'); + process.exit(1); + } + + // Load encounter state + let encounter = requireEncounterState(); + + // Resolve caster + let caster: EncounterActor; + try { + caster = findActorByIdentifier(encounter, casterIdentifierRaw); + } catch (err) { + console.error((err as Error).message); + process.exit(1); + return; + } + + // Resolve targets + const targetTokens = targetsCsv.split(',').map(s => s.trim()).filter(Boolean); + if (targetTokens.length === 0) { + console.error('No valid targets provided to --targets.'); + process.exit(1); + } + const targets: EncounterActor[] = []; + for (const tok of targetTokens) { + try { + targets.push(findActorByIdentifier(encounter, tok)); + } catch (err) { + console.error((err as Error).message); + process.exit(1); + return; + } + } + + // Remove any existing Bless tags linked to this caster across all actors + const source = `conc:${caster.id}:Bless`; + for (const actor of Object.values(encounter.actors)) { + const tags = actor.tags ?? []; + for (const t of tags) { + if (t.source === source) { + encounter = encounterRemoveActorTag(encounter, actor.id, t.id); + console.log(`Removed tag ${t.id} from ${actor.name}: "Bless"`); + } + } + } + + // Start/Update concentration entry (Bless doesn’t need a single target) + encounter = startConcentration(encounter, { + casterId: caster.id, + spellName: 'Bless', + }); + + // Add Bless tag to each target + const defaultNote = 'Add 1d4 to attack rolls and saving throws'; + for (const target of targets) { + const tag: Omit<ActorTag, 'id' | 'addedAtRound'> = { + text: 'Bless', + note: noteText ?? defaultNote, + source, + }; + encounter = encounterAddActorTag(encounter, target.id, tag); + console.log(`Added tag to ${target.name}: "Bless" [${source}]`); + } + + saveEncounter(encounter); + console.log(`Started/updated concentration: Bless (${caster.name}) → ${targets.length} target(s)`); +} + // ...rest of file

**Core:** No changes required; PR-20’s endConcentration prefix sweep handles Bless tags automatically.

**Demo & Tests (single block; copy/paste after build and install)**

*# Always run the test suite first*

pnpm test

*# PR 21 — Bless sugar (multi-target concentration helper)*

*# One-shot demo script (setup → exercise → teardown)*

*# A) Setup — load a cleric and start a fresh encounter*

pnpm dev -- character load-name "Bruni"

pnpm dev -- encounter start

pnpm dev -- encounter add pc "Bruni"

pnpm dev -- encounter add monster "Goblin" --count 3

pnpm dev -- encounter list

*# B) Cast Bless on two allies (example: Bruni + Aerin if loaded, or Bruni alone plus an ally)*

*# If you already have Aerin loaded from prior runs, you can add Aerin as a PC; otherwise bless Bruni + Goblin #1 just to demo tags.*

*# Here we'll quickly add Aerin for clarity:*

pnpm dev -- character load-name "Aerin"

pnpm dev -- encounter add pc "Aerin"

pnpm dev -- encounter list

*# Bless two allies*

pnpm dev -- encounter bless "Bruni" --targets "Bruni,Aerin"

pnpm dev -- encounter list

*# C) Recast Bless with a different target set (old Bless tags should be removed automatically)*

pnpm dev -- encounter bless "Bruni" --targets "Bruni,Goblin #1,Goblin #2" --note "Add 1d4 to attacks/saves (manual)"

pnpm dev -- encounter list

*# D) End concentration (should clear all Bless tags)*

pnpm dev -- encounter concentration end "Bruni"

pnpm dev -- encounter list

*# E) Teardown — end the encounter and unload characters*

pnpm dev -- encounter end

pnpm dev -- encounter list

pnpm dev -- character unload

If you want me to tweak the console copy, default note text, or the behavior when a target token doesn’t resolve (e.g., skip vs. fail hard), say the word and I’ll adjust. Otherwise this is ready to implement as a single combined commit for PR-21.

**You said:**

before I run that, can you get me the correct comands to save a couple more characters so we have that around? there currently isn't a bruni also, for the test commands each PR, can we format in such a way that i can paste them all in one block?

**ChatGPT said:**

Thought for 2m 0s

Love it — I saw your run and the extra character load-name + character unload bookends. 👍

Two things for you:

1. **Add a couple of named PCs (so they’re always around)**

I don’t see a “create/save character” verb in the CLI surface we used (we have character load "<path.json>", character load-name "<Name>", and character unload" in your logs). So there are **two common ways** folks “save” named characters:

**Option A — You already have JSON files for the PCs**

Put them in your characters folder (the place load-name looks up by name), then you can always load-name them later.

**One-block PowerShell (Windows)**

*# 1) Adjust these paths to where your character JSONs live right now:*

*# (If you don’t have these files yet, skip to Option B.)*

$RepoRoot = "C:\dev\grimengine"

$CharsDir = Join-Path $RepoRoot "data\characters" *# <-- if your repo uses a different dir, change this*

New-Item -ItemType Directory -Force -Path $CharsDir | Out-Null

*# Copy in your existing files (edit the source paths on the right):*

Copy-Item "C:\path\to\Bruni.json" (Join-Path $CharsDir "Bruni.json") -Force

Copy-Item "C:\path\to\Kara.json" (Join-Path $CharsDir "Kara.json") -Force

*# Quick smoke: load by name so they’re cached/ready*

pnpm dev -- character load-name "Bruni"

pnpm dev -- character unload

pnpm dev -- character load-name "Kara"

pnpm dev -- character unload

**Option B — You don’t have JSONs yet (use your file-based loader)**

If you prefer to store them outside the repo, just **load from a file path** whenever you need them. (This doesn’t register a “named” character; it just loads the session character from disk.)

**One-block PowerShell**

*# Adjust these to wherever you keep characters outside the repo:*

$BruniPath = "C:\characters\Bruni.json"

$KaraPath = "C:\characters\Kara.json"

*# Load each once to verify:*

pnpm dev -- character load "$BruniPath"

pnpm dev -- character unload

pnpm dev -- character load "$KaraPath"

pnpm dev -- character unload

If you want me to generate baseline JSONs for, say, **Bruni (cleric 3)** and **Kara (ranger 3)** that match your engine’s schema, I can spit those out and you can drop them into either location above.

1. **PR 21 demo in a single paste-all block (with tests up top and full teardown incl. unload)**

Here’s the updated **one-block** you asked for — it assumes **Option A** above (named characters "Aerin", "Bruni", "Kara" are available via load-name). If you’re using Option B, swap the load-name calls for character load "<path.json>".

# Run tests first

pnpm test

# Character prep (load, then unload so the name is ready for later)

pnpm dev *-- character load-name "Aerin"*

pnpm dev *-- character unload*

pnpm dev *-- character load-name "Bruni"*

pnpm dev *-- character unload*

pnpm dev *-- character load-name "Kara"*

pnpm dev *-- character unload*

# PR 21 — Bless sugar (multi-target concentration helper)

# One-shot demo (setup → exercise → teardown with unload)

# Setup: start encounter and add two PCs + a few foes

pnpm dev *-- encounter start*

pnpm dev *-- character load-name "Bruni"*

pnpm dev *-- encounter add pc "Bruni"*

pnpm dev *-- character unload*

pnpm dev *-- character load-name "Aerin"*

pnpm dev *-- encounter add pc "Aerin"*

pnpm dev *-- character unload*

pnpm dev *-- encounter add monster "Goblin" --count 3*

pnpm dev *-- encounter list*

# Cast Bless on two allies (Bruni + Aerin)

pnpm dev *-- encounter bless "Bruni" --targets "Bruni,Aerin"*

pnpm dev *-- encounter list*

# Recast Bless to a different set (Bruni + Kara), demonstrating retarget/sweep

pnpm dev *-- character load-name "Kara"*

pnpm dev *-- encounter add pc "Kara"*

pnpm dev *-- character unload*

pnpm dev *-- encounter bless "Bruni" --targets "Bruni,Kara" --note "Add 1d4 to attacks/saves"*

pnpm dev *-- encounter list*

# End concentration (should clear all Bless tags)

pnpm dev *-- encounter concentration end "Bruni"*

pnpm dev *-- encounter list*

# Teardown: end encounter and unload any session character

pnpm dev *-- encounter end*

pnpm dev *-- encounter list*

pnpm dev *-- character unload*

If you want me to bake the **Option A** character-folder path (e.g., data/characters/) directly into the repo and add a tiny README snippet so “drop JSON here to use load-name” is explicit, I can include that as a micro PR too.

**You said:**

so let's just do one more PR before I run any of those tests. this one will be kind of small: - add a couple more character json files in the correction location - and then just in your output for the test commands (probably mostly the same as the last one), format it in a way that I can run them as one big giant command

**ChatGPT said:**

Thought for 1m 5s

Awesome—here’s a tiny in-between PR that just adds a couple of ready-to-use characters, plus a single “paste-once” command you can run in PowerShell to exercise PR-21 after this lands.

**PR 21-prep — chore(repo): add ready-to-load PCs (Bruni, Kara)**

**Summary**

Adds two character JSONs in the **same directory your**character load-name**already resolves from** (alongside Aerin.json). These are drop-in, schema-compatible clones of your existing template, with only identity/class/level/stats adjusted:

* Bruni.json — Dwarf Cleric 3 (Life)
* Kara.json — Human Ranger 3 (Hunter)

**Motivation**

You asked for a couple more PCs to be “always around” so the Bless/HM demos don’t require ad-hoc creation.

**Files added**

* data/characters/Bruni.json
* data/characters/Kara.json

Note: I matched the field layout to your existing Aerin.json so the loader prints Loaded character <Name> (lvl 3). PB +2. If you want different ability spreads or gear, we can tweak later.

**Implementation patch (unified diffs)**

**Add Bruni (Cleric 3)**

**File:** data/characters/Bruni.json

+{ + "name": "Bruni", + "level": 3, + "class": "Cleric", + "subclass": "Life", + "race": "Hill Dwarf", + "alignment": "NG", + "abilities": { "str": 14, "dex": 10, "con": 16, "int": 10, "wis": 16, "cha": 10 }, + "proficiencyBonus": 2, + "ac": 18, + "hp": { "max": 27, "current": 27, "temporary": 0 }, + "speed": 25, + "skills": { "medicine": true, "religion": true, "insight": true, "history": false }, + "savingThrows": { "wis": true, "cha": true }, + "equipment": ["Chain Mail", "Shield", "Mace", "Holy Symbol"], + "features": ["Divine Domain (Life)", "Channel Divinity (1/rest)", "Disciple of Life"], + "spellcasting": { + "spellSaveDC": 13, + "spellAttackMod": 5, + "slots": { "1": 4, "2": 2 }, + "prepared": ["Bless", "Cure Wounds", "Guiding Bolt", "Healing Word", "Sanctuary", "Spiritual Weapon", "Aid"], + "cantrips": ["Guidance", "Sacred Flame", "Thaumaturgy"] + } +}

**Add Kara (Ranger 3)**

**File:** data/characters/Kara.json

+{ + "name": "Kara", + "level": 3, + "class": "Ranger", + "subclass": "Hunter", + "race": "Human", + "alignment": "CG", + "abilities": { "str": 12, "dex": 16, "con": 14, "int": 10, "wis": 14, "cha": 8 }, + "proficiencyBonus": 2, + "ac": 15, + "hp": { "max": 28, "current": 28, "temporary": 0 }, + "speed": 30, + "skills": { "perception": true, "stealth": true, "survival": true, "athletics": true }, + "savingThrows": { "str": true, "dex": true }, + "equipment": ["Studded Leather", "Longbow", "Shortswords"], + "features": ["Favored Enemy", "Natural Explorer", "Fighting Style (Archery)", "Hunter’s Prey (Colossus Slayer)"], + "spellcasting": { + "spellSaveDC": 12, + "spellAttackMod": 4, + "slots": { "1": 3 }, + "prepared": ["Hunter's Mark", "Cure Wounds"], + "cantrips": [] + } +}

If your engine expects any different key names (rare), I mirrored the Aerin template you’re already loading so these resolve cleanly via character load-name.

**One-liner (PowerShell) to run PR-21 after this lands**

This is the “giant single command” version you asked for. Paste it once into a PowerShell window at the repo root:

pnpm test; pnpm dev -- character load-name "Aerin"; pnpm dev -- character unload; pnpm dev -- character load-name "Bruni"; pnpm dev -- character unload; pnpm dev -- character load-name "Kara"; pnpm dev -- character unload; pnpm dev -- encounter start; pnpm dev -- character load-name "Bruni"; pnpm dev -- encounter add pc "Bruni"; pnpm dev -- character unload; pnpm dev -- character load-name "Aerin"; pnpm dev -- encounter add pc "Aerin"; pnpm dev -- character unload; pnpm dev -- encounter add monster "Goblin" --count 3; pnpm dev -- encounter list; pnpm dev -- encounter bless "Bruni" --targets "Bruni,Aerin"; pnpm dev -- encounter list; pnpm dev -- character load-name "Kara"; pnpm dev -- encounter add pc "Kara"; pnpm dev -- character unload; pnpm dev -- encounter bless "Bruni" --targets "Bruni,Kara" --note "Add 1d4 to attacks/saves"; pnpm dev -- encounter list; pnpm dev -- encounter concentration end "Bruni"; pnpm dev -- encounter list; pnpm dev -- encounter end; pnpm dev -- encounter list; pnpm dev -- character unload

* Works as a single paste/run in **PowerShell** (; separates commands).
* If you prefer **cmd.exe**, replace ; with && (and drop the couple of runs that intentionally fail with exit 1 like encounter list after end), or just use PowerShell as above.

If you want different statlines or gear for Bruni/Kara, tell me what to tweak and I’ll update the JSONs in this PR.

**ChatGPT said:**

Stopped after 12 seconds

**ChatGPT said:**

Stopped after 11 seconds

I'll set up the computer environment and inspect the project directory to identify the correct location for adding the new character JSON files. After reviewing the structure, I will ensure the files are placed appropriately and integrate the user's new Markdown formatting instructions into the final response.

Stopped

ChatGPT can make mistakes. Check important info.