v0.7.0-beta Requirements Gathering

Back to Index

Overview

The `ttrpg` package is a tabletop roleplaying game engine that allows developers to create their own rulesets through generic interfaces. The core functionality covered in this beta release will be *simple* implementations of basic needs of my own homebrew party's sessions.

Requirements outline

General

Back to top

Two core parts of making a reusable ttrpg library is configuring them via "modules", similar to how D&D itself is extended by its players. Two main pieces of functionality aid in this vision:

1. Users will have the ability to pass validators, that run as middleware functions on arguments, into any `builder`, thus defining your own rules for what is and is not a "valid" input for core structures like `Weapons` and `Characters`.
2. Users can extend data shapes through generics to implement custom attributes not provided by the core `ttrpg` engine.

Character sheets and in-game tracking

Back to top

The main functionality desired by the front-end client for a beta release is the ability to create characters, view character sheet information, and track changing values in-game.

The sheet will be generative, taking properties from a given character and populating as much of the mod-based values (i.e. skills, saving rolls, ...) as possible. Anything requiring a stored state should belong to the `Character` itself to cut down on database requirements for applications.

A snapshot of the sheet can be saved at any time for any kind of implementation: backup, sharing, etc.

Character values can be managed from the character sheet; values like `level`, `hp`, and other in-game stats that may change.

Inventory equip/store values can be managed from the character sheet.
Inventory (all) equipped items' actions the sheet's `actions` values.

Sheet requirements

- [ ] Sheet hydrates from a `Characer`
- [ ] Level can be changed from the sheet
- [ ] HP can be tracked from the sheet
- [ ] Gold can be tracked from the sheet
- [ ] Equipped items' actions are available from the sheet
- [ ] Items can be equipped and stored from the sheet

Custom Item creation

[Back to top]

To enable users of the `ttrpg` library to build their own worlds, custom items like weapons, wearables, and trinkets need to be given builders and interfaces. `Item` will be a generic interface that enforces basic requirements for *any* item, while extensions will be provided for `WeaponItem`, `WearableItem`, and `TrinketItem` types.

Trinkets: Items that are unusable but hold some perceived value, such as a pet rock or an NFT. Will be made generic to be extendable for homebrew rulesets.

Wearables: Items that are meant to be worn and optionally give benefit to the wearer. Will be made generic to be extendable for homebrew rulesets.

Weapons: Items that are meant to be used in combat and issue some form of damage or debuff. Will be made generic to be extendable for homebrew rulesets. Weapons, have limitations. A limiter should be implemented to allow a use-cost to be payed, and restricting use when the limit is reached.

All `Item` children will inherit the functionality to interact with the owner's `CharacterSheet`.

Item requirements

- [ ] Basic `Item` class that provides interaction functionality to children
- [ ] `WeaponItem` class that lets users create a basic weapon with its own actions
- [ ] `TrinketItem` class that lets users create an item with no use, only value
- [ ] `WearableItem` class that lets users create wearable items like armor that can interact with the wearer's `CharacterSheet`

Character creation

Back to top

Creating characters is integral to the tabletop games experience! Because the `ttrpg` library is meant to be extendable, Character base traits will be made generic to extend further attributes. Base traits will be pulled from common 5e character attributes like `name`, `alignment`, and the core `abilities` scores.

> In future versions, perhaps by `v1.0.0`, custom `CharacterAbilities` may be implemented. Because character sheets are generative based on the ability values (`DEX`, `STRENGTH`, etc), it'll take further extension of `CharacterSheet` generative functionality to achieve this homebrew element.

Character requirements

- [ ] Characters can defined a name (freely), race, and class (from enum)
- [ ] Characters can define physical features, albeit loosely (primitive `strign` and `number` types).
- [ ] Character can define background, personality, ideals, bonds, and flaws as `string | string[]` types (read: just *not* in any way enumerated)
- [ ] Character can set an alignment from the standard alignment chart options

Inventory functionality

Back to top

Inventories belong to a `Character`, and thus should try their best to scale non-linearly, as the `Character` itself is what users will store in a database.

Items should designated as equipped or stored, and through interfaces for the `CharacterSheet`, the equipped items *only* can be used. Stored items cannot be used.

Item flow in and out of the inventory through interfaces on the `CharacterInventory` type should be easy. Incoming items always designated as stored (not equipped).

Inventory requirements

- [ ] Inventory can take in items, scaling efficiently, and dispose of consumable-style items.
- [ ] Inventory manages a stored/equipped state
- [ ] Items with an `equipped` state can be used through the character sheet interface