Programming overview

Skills and Combat

Skill Types

There are several basic skill types. Simple abilities like Autoattack will usually only contain one of these (AoE attack), but more complex abilities can contain several. It's also possible to combine several "layers" - for example a Ground Targeted skill might have a timer, and every second that target location spawns several projectiles flying outward in a circle.

- Targeted
  - a basic skill where a certain effect is applied to an enemy.
- Projectile
  - Spawns a moving godot scene with a hitbox, applying certain effects to hit enemies
- Ground Targeted
  - spawns a godot scene at target location. Can then spawn AoE attacks or projectiles, ...
- Buff
  - Applied to target. Usually timed. Can either modify stats while it's there, or repeatedly apply some effect on a timer.
- Debuff
  - Removes one or several types of buff from target. Can be friendly or hostile, depending on the removed effect types.
- AoE Attack
  - spawns a godot scene located relative to the player, for example swinging a sword. Checks what's inside and applies effect.

These Skill Types then call an Interface for triggering a combat event on anything they hit.

Interfaces

A theoretical setup of how things could interact.

Combat Events: Incoming

To try and keep it "modular" and work with composition, we'll define several interfaces that can have effects on the targeted entity.

Damage:

We'll define incoming damage as:

- Amount (required)
- Damage Type (optional)
- Source (required, probably)

This allows us some flexibility in applying direct damage. The function that's called by this signal then calculates if there's any resistances, and logs the event in combat log.

Healing:

- Amount
- Source

Healing is a seperate interface because this makes it clearer that we can implement different functionality, like a "healing received" modifier for example when players have to protect an NPC, to not make it too easy. We want to keep this logic seperate from the damage logic.

Buff:

Buffs can be positive OR negative, since I did like the way it's done in 7 days to die - a buff adds some status effect, and debuff means the effect is removed.

- Buff ID (required) - a buff ID that we can look up in the skill database
- Duration (optional, if there is a "standard" duration in the db)
- Strength (optional) - if we implement a stat like "buff effect", we need this - likely for stuff like "burn dmg"
- Source (required, probably)

Debuff:

Removes buffs/status effects/whatever.

- Debuff Skill ID (required) - we can look up what Buff Types are removed in the skill DB
- Source (required, probably)

Movement:

Moves the entity. Can be friendly or hostile. "grappling" means a constant "force" is applied, pulling entity towards target location.

> Game Design Note: Heavily prefer both firendly and hostile movement to be skills cast by the player. Avoid situations where players get moved against their will, except maybe bossfights.

- Type (required) - Teleport, Fixed Dash, Grappling
- Target Location (optional) - Required for Teleport and Grappling types.
- Direction (optional) - A vector. Usually required for Dash.
- Speed (optional) - for dash and grappling type.
- Duration (optional) - for dash and grappling type.
- Source (required, probably)

Special:

For things that do not fit in the other categories - For example, Resurrect skills, maybe Instant Kill events that can't be resisted, ...

- Type (required)
- Source (required, probably)

Component system

We use components to avoid having to copypaste code, while also not running into inheritance issues that class systems have.
(classes are still used, but mainly to have pre-configured classes already containing components that are correctly linked to each other)

skills_component

Skills component adds the ability to cast skills to it's parent entity.

Interfaces:

cast_skill(id, target)

casts the specified skill, optionally on a given target. target can be a reference or location (? figure out if thats good, maybe look up in the skill db whether the skill is entity-target or ground target?)

get_stats()

if the skill can be affected by stats, the skill component checks if the parent has a stats component, if so it reads the relevant stats and applies them to the skill.

spawn_skill_scene(data, target)

once ready, the skills component instantiates the necessary scenes for the skill, and places them at the given target.

Neccessary initialisation:

skills_db:

the component needs a skill db to know what skills there are and which scenes to spawn for each.

skills_scenes:

we need to have pre-made scenes available for each skill. These can either be just a script, for example some buff that just modifies stats, or whole 3d scenes with animations, particles, collision boxes etc.

connecting signals:

to keep flexibility, we should implement the interfaces with signals - so skills_component needs to listen to it's parent's cast_skill signal. This also allows things like a mana component to listen to them, and we don't need to call 5 different components each skill cast. (cooldown should probably be handled by UI?)

~~player_skills_unlocked:~~

~~we need to get a list of skills that are available to the player, otherwise we'd be able to cast whatever.~~

Specifically excluded:

skills_UI:

we exclude the UI parts - mobs can cast skills but don't need this. The UI can also read data from the skills_db.

The UI will additionally only display skills the player has unlocked, so we do NOT need to check this within this component.

Classes

Mobs

Mobs are defined in one main class: mob

this mob class implements the combat component, health bars, targeting?, loot/exp drop

purposefully excluded is pathfinding