Combat Forecast Library

Overview

The Combat Forecast module is designed to help developers make more informed decisions during combat by predicting the length of encounters and the potential impact of spells. This module integrates with Sylvanas' core functionality to provide accurate combat data, enhancing strategies for PvE scenarios. Below, we'll delve into its core functions and how to effectively utilize them.

> TIP
> You should check User Combat Forecast Guide to understand what this module is about in more depth before starting to work with it.

Including the Module

Like with all other LUA modules developed by us, you will need to import the health prediction module into your project. To do so, you can just use the following lines:

```lua
---@type combat_forecast
local combat_forecast = require("common/modules/combat_forecast")
```

> WARNING
> To access the module's functions, you must use `:` instead of `.`
>
> For example, this code is not correct:
>
> ```lua
> ---@type combat_forecast
> local combat_forecast = require("common/modules/combat_forecast")
>
> local function should_cast_hard_cast_spell(player)
>     local combat_length_simple = combat_forecast.get_forecast()
>     return combat_length_simple >= 3.0
> end
> ```
>
> And this would be the corrected code:
>
> ```lua
> ---@type combat_forecast
> local combat_forecast = require("common/modules/combat_forecast")
>
> local function should_cast_hard_cast_spell(player)
>     local combat_length_simple = combat_forecast:get_forecast()
>     return combat_length_simple >= 3.0
> end
> ```

Functions

Forecast Lengths Enum

`forecast_lengths`

The forecast_lengths enum provides various lengths for combat forecasting:

- DISABLED: No forecast applied.
- VERY_SHORT: Forecast is for a very short duration.
- SHORT: Forecast is for a short duration.
- MEDIUM: Forecast is for a medium duration.
- LONG: Forecast is for a long duration.

This enum is used to specify the expected length of a combat scenario when making logic decisions.

Combat Data Retrieval

`get_forecast() -> number`

Retrieves the forecast data for the current combat situation. This function provides an overall view of the combat forecast, which can be used to adapt strategies on the fly.

`get_forecast_single(unit: game_object, include_pvp?: boolean) -> number`

Fetches the forecast data specifically for a single unit, with an option to include PvP-related considerations. This is particularly useful for predicting the impact of spells on individual targets.

Minimum Combat Length

`get_min_combat_length(forecast_mode: any, plugin_name: string, spell_name: string) -> number`

Determines the minimum combat length required for a specified forecast mode, plugin, and spell. This data helps in deciding whether to use long cooldown abilities or time-sensitive spells.

Forecast Logic Validation

`is_valid_forecast_logic(min_combat_length: number, unit?: game_object, include_pvp?: boolean) -> boolean`

Validates the forecast logic based on the minimum combat length and the specified unit. This function ensures that actions are only taken if they align with the expected duration of the encounter, avoiding the misuse of cooldowns.

Usage Example and Best Practices

Here is an example of how to implement the Combat Forecast module effectively in your code:

```lua

---@type combat_forecast
local combat_forecast = require("common/modules/combat_forecast")

local function should_cast_spell_based_on_global_forecast(spell_name)
    local min_combat_length = combat_forecast:get_min_combat_length(combat_forecast.enum.SHORT, "my_plugin", sp
    local is_valid_logic = combat_forecast:is_valid_forecast_logic(min_combat_length)

    if is_valid_logic then
        core.log("Casting " .. spell_name .. " based on combat forecast")
        return true
    else
        core.log("Skipping " .. spell_name .. " due to short combat forecast")
        return false
    end
end
```

Or, if we just want to check our main target:

```lua

---@type combat_forecast
local combat_forecast = require("common/modules/combat_forecast")

local function should_cast_spell_based_on_single_forecast(target, spell_name, forecast_max_time)
    local combat_length_single = combat_forecast:get_forecast_single(target)
    local is_valid_logic = combat_length_single <= forecast_max_time

    if is_valid_logic then
        core.log("Casting " .. spell_name .. " based on single - combat forecast")
        return true
    end

    core.log("Skipping " .. spell_name .. " due to short single - combat forecast")
    return false
end
```

Or, if we just want a quick, simple check for general usage (eg. not a very important spell)

```lua

---@type combat_forecast
local combat_forecast = require("common/modules/combat_forecast")

local function should_cast_spell_based_on_general_forecast(target, spell_name, forecast_max_time)
    local combat_length_simple = combat_forecast:get_forecast()
    local is_valid_logic = combat_length_single <= forecast_max_time

    if is_valid_logic then
        core.log("Casting " .. spell_name .. " based on single - combat forecast")
        return true
    end

    core.log("Skipping " .. spell_name .. " due to short single - combat forecast")
    return false
end
```

Best Practice Tip

> TIP
> Always ensure that you validate the combat length before casting spells with long cooldowns or spells that have a long cast time. This approach will prevent unnecessary use of critical abilities in short fights, optimizing your overall strategy.

Previous
« Spell Prediction

Next
Health Prediction Library »