PvP UI Module Library

Overview

The PvP UI Module provides the required functions for you to implement your own PvP-UI own panel. Read (pvp ui module - user) todo add link — before continuing with this guide, so you know what this module is about.

Including the Module

As with all other LUA modules developed by us, you will need to **import** the PVP Helper module into your project. To do so, you can use the following lines:

```lua
--@type ui_buttons_info
local ui_buttons_info = require("common/utility/ui_buttons_info")
```

> ⚠ WARNING
> To access the module's functions, you must use `:` instead of `.`. This is the only module where `:` is not required.

Functions

> ⓘ NOTE
> Almost everything is handled automatically, so there is just one function that you must call under any circumstance. This is the `push_button` function. All the other functions are just regarding available customizations for the user, that are centralized within the plugin and can be modified by the user directly on the window that spawns. You should take into consideration these settings that the user might modify for your logic so your plugin is coherent.

> ⚠ WARNING
> The function that you are going to pass to the GUI module MUST return TRUE at some point. When your logic function returns true, the logic is removed from the queue. Otherwise, the button will be permanently stuck on the "On Queue" state. Check the code example and the rest of documentation so you can fully understand how this works. Just keep this information in mind for when you read the "push_button" function definition, just below.

`push_button(button_id: string, title: string, spell_ids:table<number>, logic_function:fun) -> nil`

Parameters explanation:

1. button_id: This is the unique identifier for the button.
2. title: This is the name that will appear in the button. For example, if you want to use scatter and trap, a desirable name would be Scatter-Trap, for example.

> ⓘ NOTE
> Avoid long title names, as the buttons should be as small as possible so the PvP UI Window occupies as little space as possible on the screen.

3. spell_ids: These are the ids of the spells that you want to check the CD of. For example, if you want the button to be disabled when Scatter and Trap are on cooldown, you need to pass `{scatter_id, trap_id}` to this function. This is independent of the logic function, so you can just pass here Trap cooldown and then also cast Scatter.
4. function: This is the logic that will be run when the user presses the button from the UI.

> ⚠ WARNING
> This function must be called just once, on script load. Do not place it inside any callback.

`get_button_info(button_id: string) -> table`

Return value explanation: This function returns a table containing all available data for the button with the specified ID. This table has the following members:

button_id -> The id of the button.
title -> The title of the button.
spell_ids -> The table containing the IDs that are used to check the remaining time of the logic.
is_pressed -> If the button is pressed now
is_enabled -> If the button is enabled (when disabled, it won't be shown in the UI)
last_trigger_time -> Last time that the button was pressed.
is_attempting_to_run_logic -> If the logic is trying to be run right now (the logic is on queue).
arena_frame_pressed_to -> The index of the button that was pressed. This is used internally to handle the target.
logic -> The logic_function itself.

`get_current_buttons_info() -> table`

Returns the table containing all the available UI buttons information. (See the previous function to see what elements does a UI button table contain)

`is_logic_attempting_only_once() -> boolean`

Returns the configuration that the user set for the Logic Cast Mode, found within the GUI customization window. If true, the logic should be also attempted to be run once. Otherwise, you can apply your custom logics or behaviour. You should always set a maximum timer to reset the button state (return true from its function).

`get_timeout_time() -> number`

Returns the timeout time that the user set.

`.launch_checkbox -> checkbox`

> ⚠ WARNING
> This is the checkbox that you must render within your plugin's menu, so the user is able to launch the PvP GUI Window. If you don't render this button, the user won't be able to re-launch the window after they close it, or it will never be shown to begin with.

Complete Example

This is the logic that we are currently using for our Beast Mastery Hunter plugin. We made sure to explain everything in the comments, so you have an easier time understanding everything.

In the pvp_ui_implementation.lua file we have the following code:

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

--@type pvp_helper
local pvp_helper = require("common/utility/pvp_helper")

--@type spell_helper
local spell_helper = require("common/utility/spell_helper")

-- used spells ids
local inti_id = 19577
local trap_id = 187650
local scatter_id = 213691

--@type ui_buttons_info
local ui_buttons_info = require("common/utility/ui_buttons_info")

-- return true means remove logic from queue - it was casted already / aborted
local function scatter_trap_logic(local_player, target, trigger_time)
    -- target is immune to stun right now, so wait for them to stop being immune
    if pvp_helper:is_cc_immune(target, pvp_helper.cc_flags.STUN, 1000.0) then
        return false
    end

    -- check if we have pet
    local pet = local_player:get_pet()
    local is_there_pet = pet and not pet:is_dead()
    local is_trying_only_once = ui_buttons_info:is_logic_attempting_only_once()

    -- get the timeout time from the ui itself, since this can be modified by the user there
    local timeout_time = ui_buttons_info:get_timeout_time()

    local current_time = core.time()
    local attempting_time = current_time - trigger_time

    -- check if the spell has been in queue for longer than the value the user set.
    if attempting_time > timeout_time then
        return true
    end

    local tried_to_cast = false

    -- check if target is cced with at least 1 second of remaining cc time
    local is_target_cced_already, cc_flag, remaining = pvp_helper:is_crowd_controlled(target, pvp_helper.cc_fla

    if is_target_cced_already then
        -- trap is the most important spell, the other ones are just complementary to this one so they are stuc
        -- therefore, if the target is cc'ed we can just cast the trap, we don't care about stun.
        if remaining > 2500 and spell_helper:is_spell_castable(trap_id, local_player, target, true, false) then
            spell_queue:queue_spell_position(trap_id, target:get_position(), 0, "hunter MM - UI - Trap")
            -- set tried_to_cast flag to true, so if the user sets the behaviour to attempt only once we
            -- can already return true and remove this logic from the queue
            tried_to_cast = true
        end

        -- if the spell is on cooldown (> 2.0 to make sure global cooldown is not interfering) then our purpos
        -- from the queue, as we casted it already.
        return core.spell_book.get_spell_cooldown(trap_id) > 2.0
    else
        -- target was not cc'ed, which means that we have to cc him.
        -- first prio is pet stun since it doesn't share DR with trap.

        -- check if our pet is impaired
        local pet_cced = false
        if is_there_pet  then
            pet_cced, cc_flag, remaining = pvp_helper:is_crowd_controlled(pet, pvp_helper.cc_flags.ANY_BUT_SLOW
        end

        -- if we can cast intimidation (pet not cc'ed, spell castable) then we cast it
        if spell_helper:is_spell_castable(inti_id, local_player, target, false, false) and not pet_cced then
            spell_queue:queue_spell_target(inti_id, target, 0, "hunter MM - UI - Intí")
        else
            -- otherwise, ONLY if intimidation is on CD, we try to cast scatter. Otherwise we just wait for the
            -- to be able to cast the spell to the target
            if core.spell_book.get_spell_cooldown(inti_id) > 2.0 then
                if spell_helper:is_spell_castable(scatter_id, local_player, target, false, false) then
                    spell_queue:queue_spell_target(scatter_id, target, 0, "hunter MM - UI - Intí (Scatter Backu
                end
            end
        end
    end

    -- we already tried to cast, if the user set the behaviour to cast only once then this function fulfilled
    -- remove it from the queue.
    if tried_to_cast then
        tried_to_cast = is_trying_only_once
    end

    -- we always remove this spell from queue if trap is on cd (was casted) or already attempted to cast and us
    return core.spell_book.get_spell_cooldown(trap_id) > 2.0 or tried_to_cast
end

-- call this in the main, ONLY ONCE. Do not place it inside any callback.
local function set_pvp_ui_buttons()
    -- PARAMETERS explanation:
    -- 1 -> remember to always use an unique identifier for each button.
    -- 2 -> remember to use a short title (as short as possible, but the user shuld still be able to understand
    -- 3 -> the ids of the spells that will be taken into account for tue GUI for the cooldown. In this case, o
    --      to be disabled. The GUI won't care about intimidation or scatter shot.
    -- 4 -> the logic itself. Remember it MUST return TRUE for the logic to be removed from queue. If your func
    --      circumstances, the button will be stuck forever in the "On Queue" state after being pressed.
    ui_buttons_info:push_button("hunter_mm_scatter_trap", "Stun-Trap", {trap_id}, scatter_trap_logic)
end

local function hide_time_slider()
    -- if you want to use a custom timeout slider and don't want to let the user modify this value, hide the sl
    -- GUI customization window. (set the parameter to TRUE instead of FALSE).
    ui_buttons_info:set_no_render_timeout_time_slider_flag(false)
end

-- export these functions (and the launch button) to our main file, where we will call them
return
{
    set_pvp_ui_buttons = set_pvp_ui_buttons,
    launch_pvp_ui_button = buttons.launch_checkbox,
    hide_time_slider = hide_time_slider,
}
```

Then, for the main file we just have:

```lua
-- {...}
-- (this is NOT inside any callback)
local pvp_ui = require("pvp_ui_logics")

ui_buttons_info:set_pvp_ui_buttons()
ui_buttons_info:hide_time_slider()
-- {...}

-- (this IS inside the on_render_menu callback)
local on_render_menu()
    -- {...}
    pvp_ui.launch_pvp_ui_button:render("Enable PvP UI Window")
    -- {...}
end
```

This is the behaviour expected for the code above: