Inhalt

Zusätzliche Gebäudeschalter

Das Gebäudemenü wurde erweitert. Es können nun 6 weitere Buttons platziert werden. Diese Buttons können Kosten anzeigen.

Du kannst Buttons nur im lokalen Skript erstellen!

Die Priorität

Gebäudeschalter werden an eine vorgegebene Position angebracht oder frei platziert.

Es gibt 3 Ebenen der Priorität.
Buttons werden entsprechen der Priorität bis maximal 6 Stück eingeblendet.

Es gibt allgemeine Buttons, typgebundene Buttons und namensgebundene Buttons.

- Allgemeine Buttons haben die niedrigste Priorität.
- Typgebundene Buttons kommen vor allgemeinen Buttons.
- Namensgebundene Buttons kommen vor allen anderen Arten.

Die Funktionen

Du kannst Buttons mit den entsprechenden Funktionen anlegen.

Allgemeine Buttons

`API.AddBuildingButton`
`API.AddBuildingButtonAtPosition`

Typgebundene Buttons

`API.AddBuildingButtonByType`
`API.AddBuildingButtonByTypeAtPosition`

Namensgebundene Buttons

`API.AddBuildingButtonByEntity`
`API.AddBuildingButtonByEntityAtPosition`

Die Verwendung

Buttons werden durch den Aufruf der entsprechenden Funktion angelegt.

```
API.AddBuildingButton(
    ButtonActionFunction,
    ButtonTooltipFunction,
    ButtonUpdateFunction
);
```

Ein Button benötigt immer 3 Funktionen.

- Eine Funktion, die beim Klick ausgelöst wird.
- Eine Funktion, die den angezeigten Text des Tooltip steuert.
- Eine Funktion, die Sichtbarkeit und Icon des Button steuert.

Jede dieser Funktionen erhält 2 Parameter.
Die Widget-ID identifiziert den Button.
Die Entity-ID identifiziert das Gebäude.

Beispiel: Single Stop

Das individuelle Stoppen von Gebäuden ist ein beliebtes Feature. Ein Rohstoffgebäude oder ein Stadtgebäude kann einzeln angehalten werden. Dieses Beispiel behandelt die Implementation. Die Funktionen werden wie zuvor beschrieben verwendet.

Action-Funktion

Dies ist die Funktion, die beim Klick auf den Button ausgeführt wird.

```
function BuildingSingleStop_Action(_WidgetID, _EntityID)
    -- Den Gestoppt-Status abfragen
    local IsStopped = Logic.IsBuildingStopped(_EntityID);
    -- Den Gestoppt-Status ins Gegenteil umkehren
    GUI.SetStoppedState(_EntityID, not IsStopped);
end
```

Tooltip-Funktion

Dies ist die Funktion, die den Tooltip Text anzeigt.

```
function BuildingSingleStop_Tooltip(WidgetID, EntityID)
    -- Wir zeigen im Regelfall den Text für "Produktion anhalten" an.
    local Title = "Produktion anhalten";
    local Text  = "- Gebäude produziert keine Waren {cr}- Siedler "..
                  " verbrauchen keine Güter {cr}- Bedürfnisse müssen "..
                  " nicht erfüllt werden";
    -- Wenn das Gebäude gestoppt ist, wollen wir den Text für "Produktion
    -- fortführen" anzeigen.
    if Logic.IsBuildingStopped(_EntityID) then
        Title = "Produktion fortführen";
        Text  = "- Gebäude produziert Waren {cr}- Siedler verbrauchen"..
                " Güter {cr}- Bedürfnisse müssen erfüllt werden";
    end
    -- Texte an die Tooltip-Funktion übergeben
    API.SetTooltipCosts(Title, Text);
end
```

Update-Funktion

Dies ist die Funktion, die den Button steuert.

```
function BuildingSingleStop_Update(_WidgetID, _EntityID)
    -- Unter diesen Umständen darf der Button nicht zu sehen sein
    if (Logic.IsEntityInCategory(_EntityID, EntityCategories.OuterRimBuilding) == 0
        and Logic.IsEntityInCategory(_EntityID, EntityCategories.CityBuilding) == 0)
    or Logic.IsConstructionComplete(_EntityID) == 0 then
        XGUIEng.ShowWidget(_WidgetID, 0);
    else
        XGUIEng.ShowWidget(_WidgetID, 1);
    end
    -- Unter diesen Umständen darf der Button nicht klickbar sein
    if Logic.IsBuildingBeingUpgraded(_EntityID)
    or Logic.IsBuildingBeingKnockedDown(_EntityID)
    or Logic.IsBurning(_EntityID) then
        XGUIEng.DisableButton(_WidgetID, 1);
    else
        XGUIEng.DisableButton(_WidgetID, 0);
    end
    -- Icon setzen
    SetIcon(_WidgetID, {4, 13});
    if Logic.IsBuildingStopped(_EntityID) then
        SetIcon(_WidgetID, {4, 12});
    end
end
```