Inhalt

Verwendung

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 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 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 Installation

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: Gebäude einzeln stoppen

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
```

Funktionsaufruf

Du kannst den Button ohne Position im Menü anbringen. Dann erscheint er an der ersten verfügbaren Position.

```
API.AddBuildingButton(
    BuildingSingleStop_Action,
    BuildingSingleStop_Tooltip,
    BuildingSingleStop_Update
);
```

Alternativ kannst Du die Position angeben.

Das ist dann gut, wenn Dir nicht egal ist, wo der Button erscheint.

```
API.AddBuildingButtonAtPosition(
    355, 168,
    BuildingSingleStop_Action,
    BuildingSingleStop_Tooltip,
    BuildingSingleStop_Update
);
```

Die Position ist neben dem Belagerungsturm der Belagerungswaffenwerkstatt. Es wird kein anderer Button durch diesen Button verdeckt.

Action-Funktion

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

Beispiel: Ausbaustufen abreißen

Zwar kann man Gebäude ausbauen, aber den Ausbau nicht wieder zurücknehmen. Das folgende Beispiel implementiert den „Rückbau“ von Gebäuden. Der Spieler kann Ausbaustufen von Rohstoff- und Stadtgebäuden wieder abreißen.

Action-Funktion

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

```
function BuildingDowngrade_Action(_WidgetID, _EntityID)
    -- Gebäudestufe im globalen Skript abreißen
    -- (Funktioniert nicht im Multiplayer!)
    GUI.SendScriptCommand(string.format([[
        local ID = %d;
        local MaxHealth = Logic.GetEntityMaxHealth(ID);
        local Health = Logic.GetEntityHealth(ID);
        if Health >= (MaxHealth - 200) then
            Logic.HurtEntity(ID, Health - (MaxHealth - 200));
        end
    ]], _EntityID));
end
```

Alle Rohstoff- und Stadtgebäude haben 200, 400 und 600 Gesundheit – je nach Ausbaustufe. Du kannst das ausnutzen und so hässliche Ruinen der zerstörten Ausbaustufe vermeiden. Das Gebäude hat sofort wieder volle Gesundheit und es muss nicht repariert werden.

Tooltip-Funktion

Dies ist die Funktion, die den Tooltip Text anzeigt.

```
function BuildingDowngrade_Tooltip(_WidgetID, _EntityID)
    -- Dies ist der Text des Tooltip.
    local Title = "Gebäude zurückbauen";
    local Text  = "- Eine Ausbaustufe wird zerstört {cr}- Der überschüssige "..
                  " Siedler wird entlassen";
    -- Dieser Text wird verwendet, wenn der Button deaktiviert ist.
    -- (Wir verwenden hier einen Standardtext des Spiels.)
    API.SetTooltipCosts(Title, Text, "UI_ButtonDisabled/B_SiegeEngineWorkshop");
end
```

Update-Funktion

Dies ist die Funktion, die den Button steuert.

```
function BuildingDowngrade_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
    or Logic.GetUpgradeLevel(_EntityID) == 0 then
        XGUIEng.ShowWidget(_WidgetID, 0);
    else
        XGUIEng.ShowWidget(_WidgetID, 1);
    end
    -- Unter diesen Umständen darf der Button nicht klickbar sein
    if Logic.GetEntityMaxHealth(_EntityID) > Logic.GetEntityHealth(_EntityID)
    or 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, {3, 15});
end
```

Funktionsaufruf

Du kannst den Button ohne Position im Menü anbringen. Dann erscheint er an der ersten verfügbaren Position.

```
API.AddBuildingButton(
    BuildingDowngrade_Action,
    BuildingDowngrade_Tooltip,
    BuildingDowngrade_Update
);
```

Natürlich kannst Du auch hier die Position bestimmen.

```
API.AddBuildingButtonAtPosition(
    215, 62,
    BuildingDowngrade_Action,
    BuildingDowngrade_Tooltip,
    BuildingDowngrade_Update
);
```

Die Position ist neben dem Upgrade-Button. Passt thematisch ganz gut zusammen.

Beispiel: Söldner in der Burg

Ein komplexeres Beispiel gefällig? Ja?

Hier werden Söldner in der Burg angeboten, sobald diese einmal ausgebaut wurde.

Action-Funktion

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

```
function BuildingMeleeMercenary_Action(_WidgetID, _EntityID)
    -- Wir verwenden die Basispreise und machen sie teurer.
    -- Dafür bleibt der Preis stabil.
    local Type = Entities.U_MilitaryBandit_Melee_NE;
    local Price = MerchantSystem.BasePrices[Type] * 3;

    -- Gold des Spielers prüfen
    -- (Wir verwenden Standardtexte für die Meldung.)
    if GetPlayerResources(Goods.G_Gold, GUI.GetPlayerID()) < Price then
        local TextKey = "Feedback_TextLines/TextLine_NotEnough_G_Gold";
        local TextMsg = XGUIEng.GetStringTableText(TextKey);
        API.Message(TextMsg);
        return;
    end
    -- Soldatenanzahl prüfen
    -- (Wir verwenden Standardtexte für die Meldung.)
    local SoldierNow = Logic.GetCurrentSoldierCount(GUI.GetPlayerID());
    local SoldierMax = Logic.GetCurrentSoldierLimit(GUI.GetPlayerID());
    if SoldierNow > SoldierMax-3 then
        local TextKey = "Feedback_TextLines/TextLine_NotEnoughRoomToBuyMilitary";
        local TextMsg = XGUIEng.GetStringTableText(TextKey);
        API.Message(TextMsg);
        return;
    end

    -- Einheit als Script Command erzeugen
    -- (Funktioniert nicht im Multiplayer!)
    API.Play2DSound("ui/menu_left_gold_pay");
    GUI.SendScriptCommand(string.format([[
        local ID = %d;
        local PlayerID = %d;
        local Orientation = Logic.GetEntityOrientation(ID) -90;
        local x,y = Logic.GetBuildingApproachPosition(ID);
        Logic.CreateBattalion(%d, x, y, Orientation, PlayerID, 3);
        AddGood(Goods.G_Gold, (-1) * %d, PlayerID);
    ]], _EntityID, GUI.GetPlayerID(), Type, Price))
end
```

Tooltip-Funktion

Dies ist die Funktion, die den Tooltip Text anzeigt.

```
function BuildingMeleeMercenary_Tooltip(_WidgetID, _EntityID)
    -- Wir verwenden die Basispreise und machen sie teurer.
    -- Dafür bleibt der Preis stabil.
    local Type = Entities.U_MilitaryBandit_Melee_NE;
    local Price = MerchantSystem.BasePrices[Type] * 3;
    -- Dies ist der Text des Tooltip.
    -- (Wir verwenden die Standardtexte für den Söldnerkauf.)
    API.SetTooltipCosts(
        "UI_ObjectNames/HireMercenaries",
        "UI_ObjectDescription/HireMercenaries",
        "UI_ButtonDisabled/B_SiegeEngineWorkshop",
        {Goods.G_Gold, Price}
    );
end
```

Update-Funktion

Dies ist die Funktion, die den Button steuert.

```
function BuildingMeleeMercenary_Update(_WidgetID, _EntityID)
    -- Unter diesen Umständen darf der Button nicht klickbar sein
    if Logic.IsBuildingBeingUpgraded(_EntityID)
    or Logic.GetUpgradeLevel(_EntityID) < 1 then
        XGUIEng.DisableButton(_WidgetID, 1);
    else
        XGUIEng.DisableButton(_WidgetID, 0);
    end
    -- Icon setzen (Axtkämpfer Icon)
    SetIcon(_WidgetID, {9, 13});
end
```

Da dieser Button nur für die Burg angezeigt wird, entfallen einige Prüfungen.

Funktionsaufruf

Du kannst den Button ohne Position im Menü anbringen. Dann erscheint er an der ersten verfügbaren Position.

Der Button soll nur in der Burg zusehen sein. Daher verwendest Du die Funktion für spezielle Gebäudetypen. Der Button kann auch hier automatisch positioniert werden.

```
API.AddBuildingButtonByType(
    -- Der Typ der Burg muss ggf. angepasst werden
    Entities.B_Castle_ME,
    BuildingMeleeMercenary_Action,
    BuildingMeleeMercenary_Tooltip,
    BuildingMeleeMercenary_Update
);
```

Natürlich gibt es auch ein Gegenstück mit Positionsangabe.

```
API.AddBuildingButtonByTypeAtPosition(
    -- Der Typ der Burg muss ggf. angepasst werden
    Entities.B_Castle_ME,
    300, 168,
    BuildingDowngrade_Action,
    BuildingDowngrade_Tooltip,
    BuildingDowngrade_Update
);
```

Die Position ist neben dem Button des Diebes.