

Doc Part 3: Integrate puzzle in your project

Table of contents:

Part 3: Read First

[link](#)

Window tab: Puzzle Creator (w_PuzzlesCreator) (overview)

[link](#)

GlobalPuzzleManager

[link](#)

Overview:

[link](#)

Manage puzzle inputs:

[link](#)

Call methods when puzzle start or stop:

[link](#)

Other options:

[link](#)

Scripting: Useful methods or variables

[link](#)

ScenePuzzleManager

[link](#)

Overview:

[link](#)

Methods call when scene starts:

[link](#)

Scripting: Useful methods or variables

[link](#)

Save system overview

[link](#)

ObjIsActivated

[link](#)

Overview:

[link](#)

How to setup an object as **ObjIsActivated** (Tuto).

[link](#)

Save Extension (script needed):

[link](#)

The Character

[link](#)

Debugger

[link](#)

Troubleshooting

[link](#)

Scripting

[link](#)

How to create a boolean method

[link](#)

Inputs: Switch between inputs types at runtime:

[link](#)

Inputs: Access inputs at runtimes

[link](#)

Useful Methods or variables for AR:

[link](#)

How to add a method (overview):

[link](#)

Part 3: Read First

This section will help you find the information you are looking for in Doc Part 3.

Window tab: Puzzle Creator

[link](#)

The window `w_PuzzleCreator` allows:

- To generate puzzles in your project.
- To manage global puzzles settings (Inputs choose, Save format)
- To create basic elements when you integrate the puzzle system into an existing project.

GlobalPuzzleManager

[link](#)

The `GlobalPuzzleManager` allows:

- To manage the player inputs when the player interacts with a puzzle.
- To manage what happens when a player enters or leaves a puzzle.

For example:

- Block the movement of the character when he enters a puzzle.
 - Allow the character's movement when he comes out of a puzzle.
 - Activate an icon, a sound when the character enters a puzzle
 - If the player holds a weapon in his hand, disable that weapon when the character enters a puzzle.
- You can add your own conditions to fits your game.

ScenePuzzleManager

[link](#)

The `ScenePuzzleManager` allows:

- To save the puzzles present in the scene.
- To initialize the puzzles at the beginning of the scene.

For example:

- Load the puzzles settings present in the scene
- Check that puzzles are not paused in the scene
- You can also put your own conditions.

Save System overview

[link](#)

A small guide to understand the save system.

ObjIsActivated

[link](#)

The `ObjIsActivated` script associated with the `SaveData` script allows:

- To save the parameters contained in an object different from the puzzles.
- To load the object settings when the scene is loading.

For example:

- It is possible to save that a door is open after solving a puzzle.
- It is possible to save / load the position of an object.

The Character:

[link](#)

A small guide to help you integrate the character provided in the asset in your project.

Debugger

[link](#)

This object help you debug puzzles more easily:

Troubleshooting

[link](#)

This section references a number of common issues.

Scripting

[link](#)

This section contains useful elements for programming:

- How to access inputs using a script.
- How to switch from one type of input to another using scripting.
- How to create a boolean method.

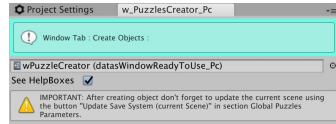
Window tab: w_PuzzlesCreator

The window **w_PuzzleCreator** allows:

- To generate puzzles in your project.
- To manage global puzzles settings (Inputs choose, Save format)
- To create basic elements when you integrate the puzzle system into an existing project.

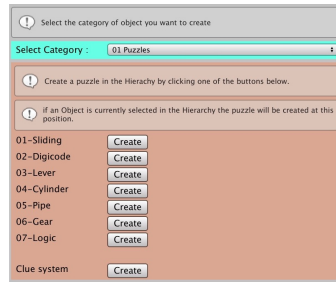
To access that window:

-Go to **Tools → Puzzles → Puzzles Creator (w_PuzzlesCreator)**



This section allows to create puzzles:

(More info about puzzle creation in [Doc Part 2](#))



Global Puzzles Parameters:

This section allows to switch between the different Puzzle input types.

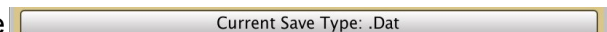
You can choose input type when the game starts.

It is possible to choose between:

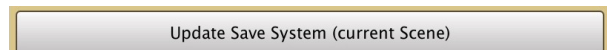
- Keyboard + Mouse
- Gamepad
- Mobile



This button allows to choose if the save system use **.Dat** format or **PlayerPrefs** format.



The button **Update Save System (current Scene)** allows to update the save system after adding a new puzzle in the a scene.



Show .Dat In Explorer open the folder that contains the save files in your computer.



Delete all save datas (all Scenes) delete all the save datas in the project.

Layer Options:

This section is useful if you want to change the default layers used by the puzzle system.

(more info about how to change a layer in [Doc Part 1 Tuto 2 Step 3: Setup Layers](#))

Layers Options	
Layer: Puzzle:	<input type="text" value="15"/>
Layer: PuzzleFeedbackCam:	<input type="text" value="16"/>
Layer: PuzzleRay:	<input type="text" value="19"/>
Layer: PuzzleDragAndDrop:	<input type="text" value="20"/>

Starter Kit:

Add Starter kit to the current scene:

This button is used when you want to integrate the puzzle system inside your project. (

more info about how to use the starter Kit in [Doc Part 1 Tuto 2 Step 4: Setup the starter kit](#))

Starter Kit
<input type="button" value="Add Starter kit to the current scene"/>
<input type="button" value="Create Inputs for Gamepad"/>

Create Inputs for Gamepad:

If you want to use gamepad in your game:

This button create two new inputs in the Inputs Manager.

The new inputs manage the gamepad mouse cursor (Vertical and Horizontal axis) in a puzzle using [Focus Mode](#).

Global Puzzle Manager:

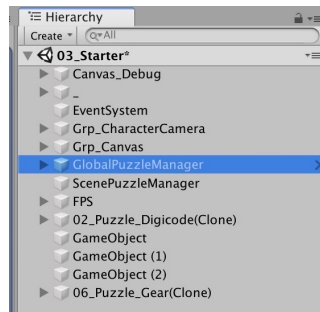
Overview:

The **GlobalPuzzleManager**:

- Manage the player inputs when the player interacts with a puzzle.
- Manage what happens when a player enters or leaves a puzzle.

For example:

- Block the movement of the character when he enters a puzzle.
 - Allows the character's movement when he comes out of a puzzle.
 - Activate an icon, a sound when the character enters a puzzle
 - Disable an icon, a sound when the character comes out of a puzzle.
 - If the player holds a weapon in his hand, disable that weapon when the character enters a puzzle.
- You can add your own conditions to fits your game.



Important:

You must have only one **globalPuzzleManager** in your scene. This object is not destroy when a new scene is loaded.
Do not rename this object in the Hierarchy.

Manage puzzle inputs:

In section **Input List** you can choose input type when the game starts (spot 1).

It is possible to choose between:

- Keyboard + Mouse
- Gamepad
- Mobile

The selected input type is green (spot 2).

(**Info:** In the example **Keyboard + Mouse** is the selected input when the game starts)



Scripting It is possible to change the Input type at runtime.

For Keyboard + Mouse call:

```
AP_GlobalPuzzleManager.instance.  
AP_SwitchPuzzleInputsToKeyboardAndMouse(true);
```

For Gamepad call:

```
AP_GlobalPuzzleManager.instance.  
AP_SwitchPuzzleInputsToGamepad(true);
```

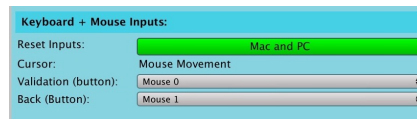
For Mobile call:

```
AP_GlobalPuzzleManager.  
instance.AP_SwitchPuzzleInputsToMobile();
```

Manage Keyboard + Mouse inputs:

In section **Keyboard + Mouse Inputs** you can choose the input when the player press:

- The button to validate an action in a puzzle.
- The button to leave a puzzle using Focus mode.



Scripting It is possible to change the Input type at runtime.

-Validation (button): (KeyCode). Call:

AP_GlobalPuzzleManager.instance.validationButtonKeyboard

-Back (button): (KeyCode). Call:

AP_GlobalPuzzleManager.instance.backButtonKeyboard

Manage Gamepad inputs:

In section **Gamepad** you can choose the inputs when the player:

- Move the cursor Horizontally
- Move the cursor Vertically
- Press the button to validate an action in a puzzle.
- Press the button to exit a puzzle on Focus mode.



Scripting It is possible to change the Input at runtime.

-Horizontal Axis: (string). Call:

AP_GlobalPuzzleManager.instance.HorizontalAxisJoystickLeft

-Vertical Axis: (string). Call:

AP_GlobalPuzzleManager.instance.VerticalAxisJoystickLeft

-Validation (button): (KeyCode). Call:

AP_GlobalPuzzleManager.instance.validationButtonJoystick

-Back (button): (KeyCode). Call:

AP_GlobalPuzzleManager.instance.backButtonJoystick

Call methods when puzzle start or stop:

It is possible to call custom methods when a puzzle starts or stops.

For example: When a puzzle starts using **focus mode** the character must stop moving in the scene. Then when the player exit the puzzle the character needs to move again.

How to do that?

-In the section **Methods call when enter a puzzle:**
Add a custom method that stop the player movement.

-In the section **Methods call when exit a puzzle:**
Add a custom method that allows the player to move

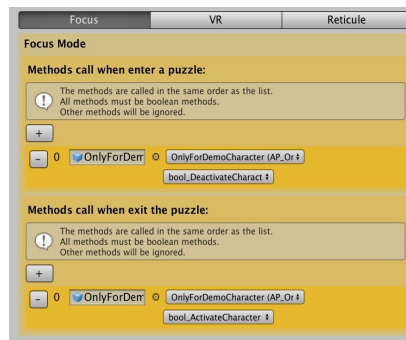
Important:

-The methods called need to be **boolean** methods.
Other methods will be ignored.

-You can add more than one boolean method.

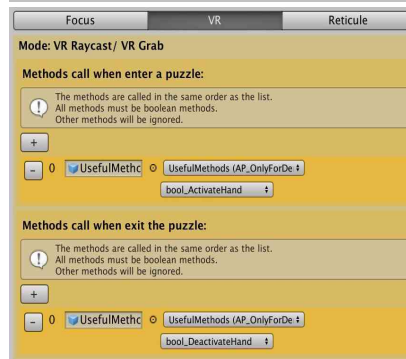
Focus Tab:

Methods used when the player enter or exit a puzzle in **Focus Mode**



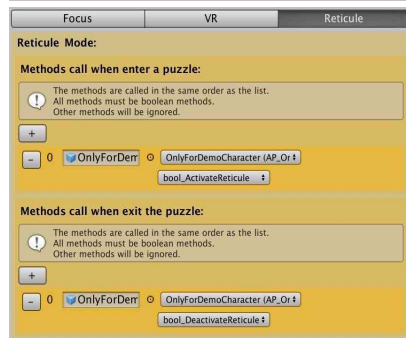
VR Tab:

Methods used when the player enter or exit a puzzle in **Mode: VR Raycast or VR Grab**



Reticule Tab:

Methods used when the player enter or exit a puzzle in **Reticule Mode**



Add new method in section **Methods call when enter a puzzle** or **Methods call when exit a puzzle** (Tuto)

Even though the sections are different, the process for adding methods remains the same.

In several sections of the puzzle system it is possible to add custom methods.

(**More Info:** If you add a method for the first time and want more information have a look to the section [How to add a method \(overview\)](#) [link](#))

Caution:

Methods call when enter a puzzle:

- Only **boolean methods** are allowed.
- Other methods are ignored.

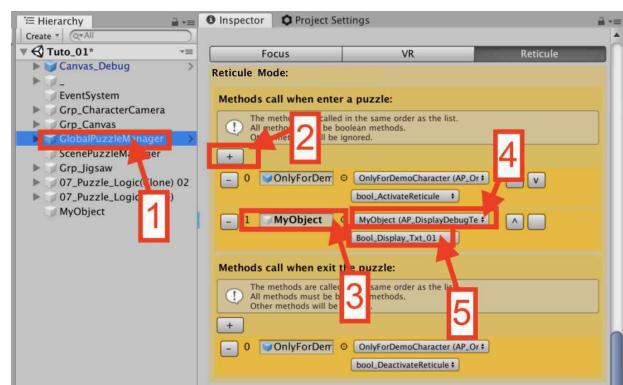
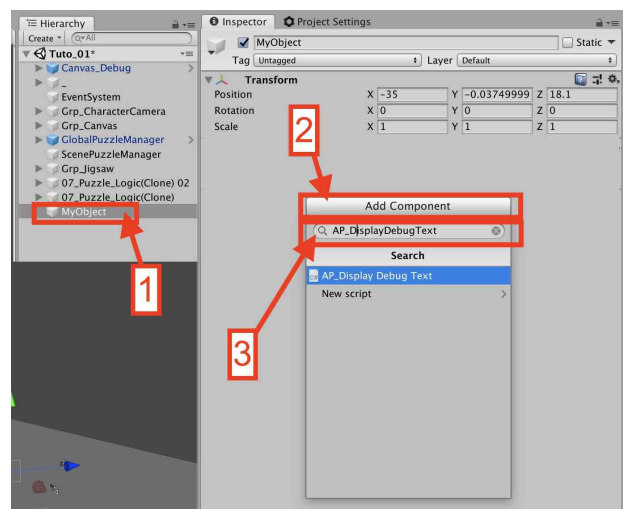
Methods call when exit a puzzle:

- Only **boolean methods** are allowed.
- Other methods are ignored.

In this example we are going to add a method in section **Methods call when enter a puzzle** in **Focus Mode**. You can follow the same process for **Methods call when exit a puzzle** in **Focus**, **VR**, and **Reticule Mode**

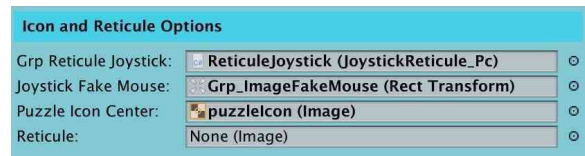
Example: Add the Method **bool_Display_Txt_01** that:

- return true.
- Display text “Activated” in the Console Tab.
- Create an empty object in the Hierarchy and rename it **MyObject** for example (spot 1).
- In the Inspector press button **Add Component** (spot 2)
- In search field write **AP_DisplayDebugText** (spot 3)
- Press keyboard key **Enter** to add the component.
- In the Hierarchy select object **GlobalPuzzleManager** (spot 1).
- In the Inspector press button **+** in the section **Method call when enter a puzzle** to create a new empty slot (spot 2)
- Drag and drop **MyObject** in the empty slot (spot 3)
- Select the script **AP_DisplayDebugText** (spot 4)
- Select the method **AP_Display_Txt** (spot 5)



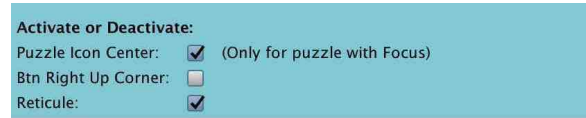
Icon and reticule Options

They are already setup. There is no reason to modify them.



Activate or deactivate:

This section allows to activate or deactivate UI elements when the player enter or leave a puzzle.



Puzzle Icon center: Allows to display a puzzle icon in the center of the screen when a puzzle is detected. This option works only on **Focus Mode**.

Btn Right Up Corner: Allows to display a puzzle icon in the right up corner of the screen when a puzzle is detected.

Reticule: Allows to activate the reticule when no puzzle is detected.

Other options:

Parent Drag And Drop is already setup. There is no reason to modify them.

CursorLockMode:

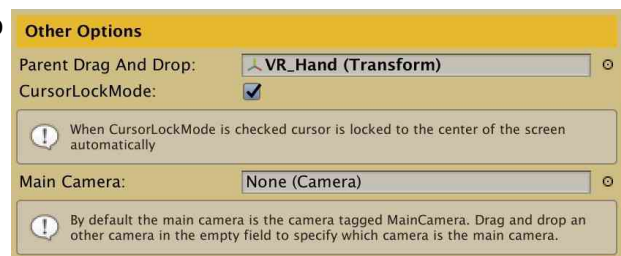
If the button is checked:

When the player leaves a puzzle the cursor is automatically locked to the center of the screen.

If the button is unchecked:

You can manage manually the cursor locked state when the player leaves a puzzle.

Note: Even if you manage manually the cursor lock state: The cursor is automatically set to confined when the player enter in a puzzle set to **Focus Mode**.



Main Camera:

If your player camera is not tagged **MainCamera**: Drag and drop your Player camera here

Scripting: Useful methods or variables that can be called at runtime:

-b_Pause (bool):

Example: If we don't want the player be able to interact with the puzzle when the game is paused.

b_Pause allows to deactivated player interactions.

if b_Pause = true paused all the puzzle

if b_Pause = false interaction with puzzle is available

Scripting It is possible to change the **b_Pause** at runtime.

b_Pause: (call) AP_GlobalPuzzleManager.instance.b_Pause

ScenePuzzleManager:

Overview:

The **scenePuzzleManager** allows:

- To initialize the puzzles at the beginning of the scene.
- To save the puzzles present in the scene.

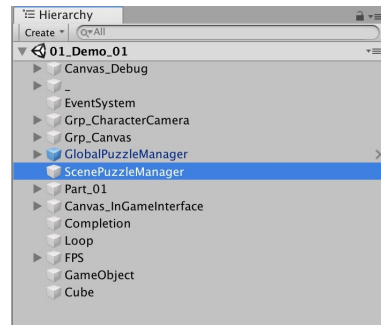
For example:

- Load the puzzles settings present in the scene
- You can also put your own conditions.

Important:

You must have only one **scenePuzzleManager** in your scene.

Do not rename this object in the Hierarchy.



Methods call when scene starts:

This section is used to call the methods contained in **methods when the scene starts**.

By default:

The parameter **Auto** (spot 1) is checked (true):

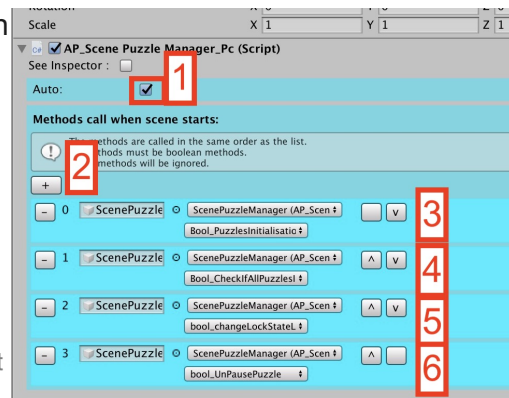
(Methods contained in section **Methods call when scene starts** (spot 2) are called when the scene starts)

if **Auto** is unchecked methods contained in section

Methods call when scene starts are not called when the scene starts.

It is possible to call **Methods call when scene starts** manually at runtime. Call the method:

StartCoroutine(CallAllTheMethodsOneByOne());
contained in the script **AP_ScenePuzzleManager_Pc**
attached to **ScenePuzzleManager** in the Hierarchy.



By default:

Methods call in section

methods when the scene starts are:

Method: **Bool_PuzzleInitialisation()**

-The puzzles are initialized (spot 3).

Method: **Bool_CheckIfAllPuzzlesInitialized()**

-Wait until all the puzzles are initialized (spot 4).

Method: **Bool_ChangeLockStateLock()**

-The Mouse cursor is locked to the center of the screen (spot 5).

Method: **Bool_UnPausePuzzle()**

-Unpause all the puzzles (spot 6).

Info: the following four methods are available in script

AP_ScenePuzzleManager_Pc

attached to **ScenePuzzleManager** in the Hierarchy

Add new method in section **Methods call when scene starts** (Tuto):

Even though the sections are different, the process for adding methods remains the same.

In several sections of the puzzle system it is possible to add custom methods.

(More Info: If you add a method for the first time and want more information have a look to the section [How to add a method \(overview\)](#) [link](#))

Caution:

Methods call when scene starts:

-Only **boolean methods** are allowed.
Other methods are ignored.

Example: Add the Method **bool_Display_Txt_01** that:

-return true.
-Display text “**Activated**” in the Console Tab when the scene starts.

-Create an empty object in the Hierarchy and rename it **MyObject** for example (spot 1).

-In the Inspector press button **Add Component** (spot 2)

-In search field write **AP_DisplayDebugText** (spot 3)

-Press keyboard key **Enter** to add the component.

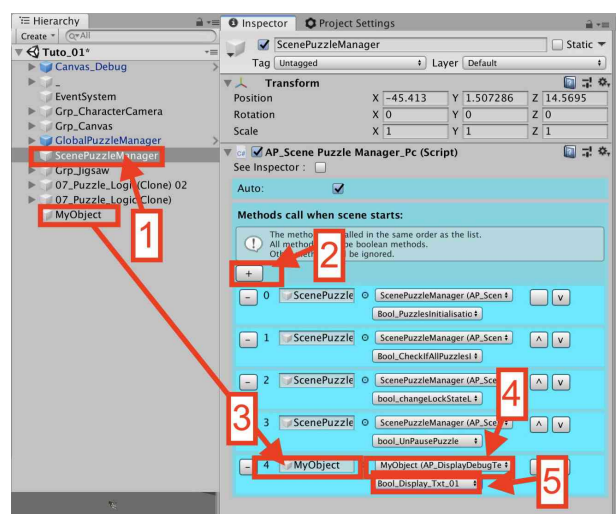
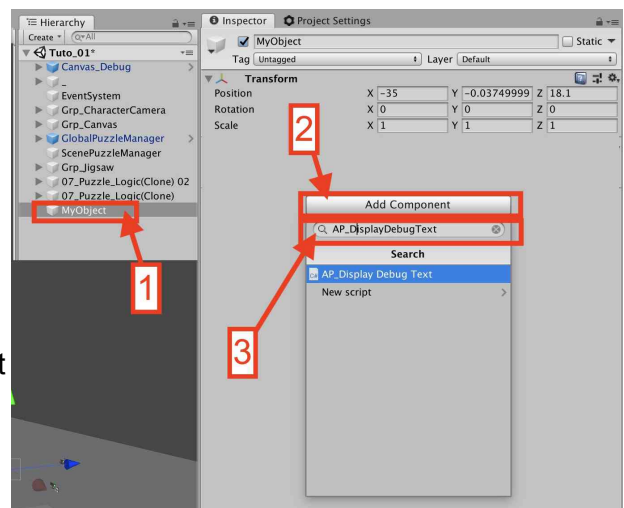
-In the Hierarchy select object **ScenePuzzleManager** (spot 1).

-In the Inspector press button **+** in the section **Method call when scene starts** to create a new empty slot (spot 2)

-Drag and drop **MyObject** in the empty slot (spot 3)

-Select the script **AP_DisplayDebugText** (spot 4)

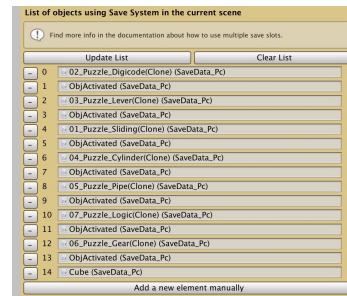
-Select the method **AP_Display_Txt** (spot 5)



List of Objects using Save System in the current scene.

This section show the list of the object integrated to the save system.

There is nothing to do in this section



Scripting: Useful methods or variables that can be called at runtime:

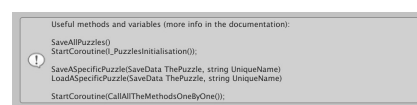
It is possible to:

Save all the puzzle manually at runtime.

Call:

`SaveAllPuzzles();`

contained in the script `AP_ScenePuzzleManager_Pc` attached to `ScenePuzzleManager` in the Hierarchy.



Initialized all the puzzles manually using the save datas at runtime.

Call:

`StartCoroutine(I_PuzzlesInitialisation());`

contained in the script `AP_ScenePuzzleManager_Pc` attached to `ScenePuzzleManager` in the Hierarchy.

Save System overview

We will take an example:

-The player approaches a puzzle.

-He solves the puzzle.

-A door opens.

The player save his game progression.

How it works:

During save process, the state of the puzzle will be automatically saved.

On the other hand, the door is an element not included in the puzzle.

During the next save, the state of the door will not be saved.

To save the state of the door, it is necessary to use the save extension system.

In summary:

During save process, the state of a puzzle is automatically saved. This includes the position of the puzzle pieces and whether the puzzle is solved or not.

On the other hand, all elements not included in the puzzle must be saved with the save expansion system

[\(more info about how to save an object different from a puzzle Link\).](#)

Other info:

Each scene has its own save file.

If the door and the puzzle are in the same scene:

They will be saved in the same file.

If you want to manually load or save go to the section [ScenePuzzleManager \(script needed\)](#)
[Link](#)

ObjIsActivated

Overview:

In the documentation we use the name

ObjIsActivated for:

-An object with the 2 specific scripts attached to it:

IsObjectActivated_Pc.cs and **SaveData_Pc.cs**

[\(More info to setup an Object as **ObjIsActivated**\)](#)

Why use **ObjIsActivated**:

For example:

Save that a door is open after solving a puzzle.

Save the position of an object.

Save if an object is enabled or disabled in the scene

ObjIsActivated allows you to save any type of parameters.

ObjIsActivated is separated in 3 parts:

First part:

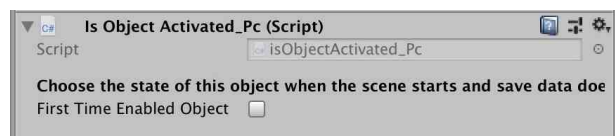
We will take an example to explain this part:

There is a key on a table.

When the player grab this key, the key disappears (disable in the Hierarchy).

During the saving process, it saved that the object is disabled in the Hierarchy.

If the player reloads the scene, the save datas indicate that the object must be disabled in the Hierarchy.



Important:

When the player loads the scene for the first time:

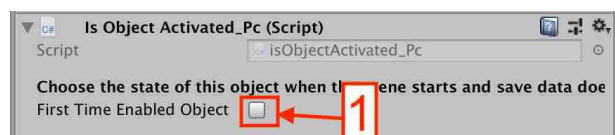
It must be indicated if the key must be enabled or disabled in the scene.

If we want the key to be enabled:

Check **FirstTimeEnabledObject** (spot 1)

If we want the key to be disabled:

Uncheck **FirstTimeEnabledObject** (spot 1)



Note: During the loading process

If a **meshRenderer** is attached to your **ObjIsActivated**:

This mesh component is enabled/disabled in the Hierarchy.

If **No meshRenderer** is attached to your **ObjIsActivated**:

This gameObject is enabled/ disabled in the Hierarchy.

Second part:

During the loading process it is possible to call custom methods.

If the object has been saved as enabled during the save process:

The methods in section

Methods call when the object needs to be activated are called.

If the object has been saved as disabled during the save process:

The methods in section

Methods call when the object needs to be deactivated are called.

[\(More about adding your custom method here\)](#)

The screenshot shows a yellow-themed interface titled "Call Methods during loading Process". It contains two sections: "Methods call when the object needs to be activated:" and "Methods call when the object needs to be deactivated:". Each section has a warning icon and a text box stating: "The methods are called in the same order as the list. All methods must be boolean methods. Other methods will be ignored." Below each text box is a small button with a "+" sign.

Third section:

This section allows to save / load your custom settings.

For example:

- Save if a door is open / closed.
- Save if a padlock is open or closed.
- Save the position of an object.

[\(More about adding custom methods on section Save Extension\)](#)

The screenshot shows a light blue-themed interface titled "Save Extension". It contains two sections: "Method call when game is saved:" and "Method call during Loading Process:". Each section has a warning icon and a text box stating: "Method must be string method. Other method will be ignored." and "Methods must be void method with a string argument. Other method will be ignored." respectively. Below each text box is a small button with a "+" sign.

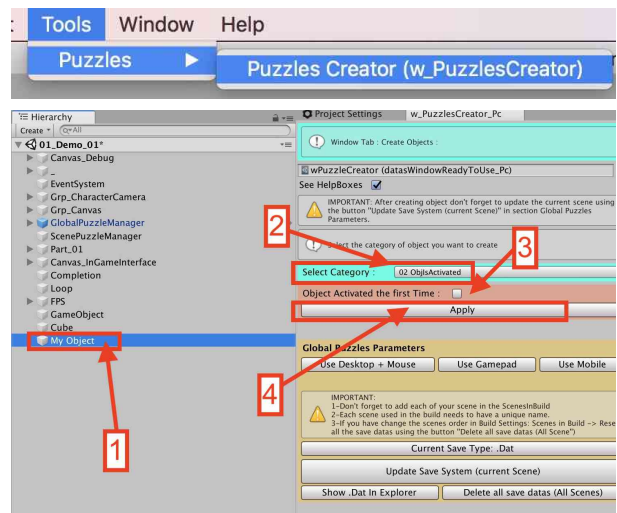
How to setup an object as **ObjIsActivated** (Tuto).

-Go to **Tools** → **Puzzles** → **Puzzles Creator** (w_PuzzlesCreator)

-In the **Hierarchy** select the object to setup (spot 1).

-In window tab **w_PuzzlesCreator_Pc** select category **02 ObjIsActivated** (spot 2).

-Choose if the object is **enabled** or **disabled** the first time the object is loaded in a scene (spot 3).



Caution:

If a **meshRenderer** is attached to your object:
This mesh component is enabled/disabled in the Hierarchy.

If **No meshRenderer** is attached to your object:
This **gameObject** is enabled/disabled in the Hierarchy.

-Click **Apply** (spot 4)

Info: In the Inspector **IsObjectActivated_Pc.cs** and **SaveData_Pc.cs** scripts have been added.

Add new method in section

Call Methods during loading process (Tuto):

Important: If you do not have an object setup as **ObjIsActivated** first read the previous section.

(How to setup an object as **ObjIsActivated**)

Even though the sections are different, the process for adding methods remains the same.

In several sections of the puzzle system it is possible to add custom methods.

(**More Info:** If you add a method for the first time and want more information have a look to the section **How to add a method (overview)**)

Caution:

In section **Call Methods during loading process:**

-Only **boolean methods** are allowed.

Other methods are ignored.

Example:

Add the Method **bool_Display_Txt_01** that:

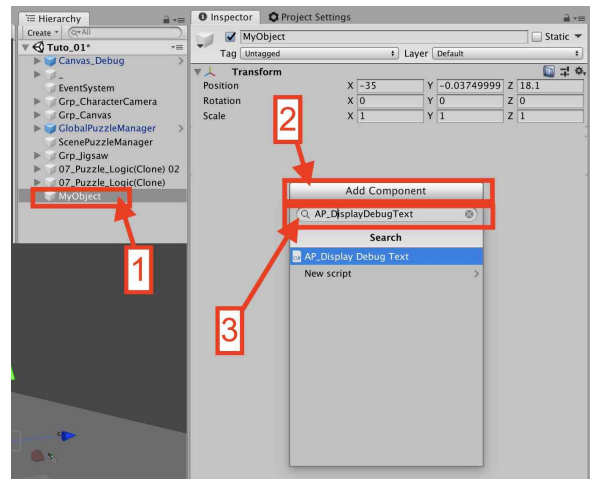
- Return true.
- Display text “**Activated**” in the Console Tab during the loading process.

-Create an empty object in the Hierarchy and rename it **MyObject** for example (spot 1).

-In the Inspector press button **Add Component** (spot 2)

-In search field write **AP_DisplayDebugText** (spot 3)

-Press keyboard key **Enter** to add the component.



-In the Hierarchy select your **ObjIsActivated** (spot 1).

-Check the **boxFirstTimeEnabledObject** (spot 2)

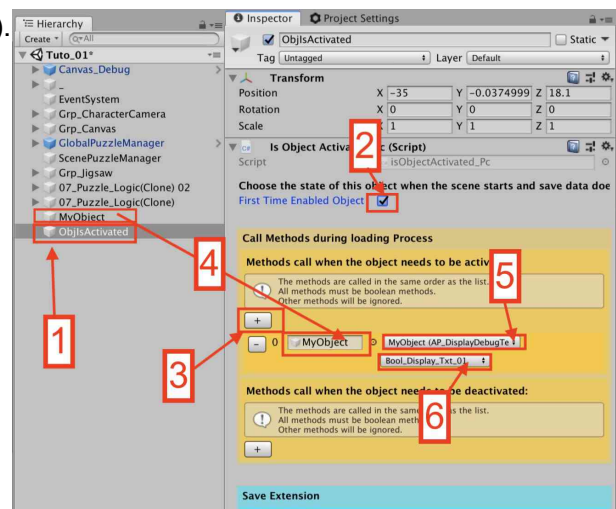
*(More info about **boxFirstTimeEnabledObject**)*

-In the Inspector press button **+** in the section **Method call when the object need to be activated** to create a new empty slot (spot 3)

-Drag and drop **MyObject** in the empty slot (spot 4)

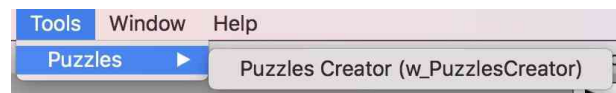
-Select the script **AP_DisplayDebugText** (spot 5)

-Select the method **AP_Display_Txt** (spot 6)



-Open the window tab **w_PuzzlesCreator**

Tools → Puzzles → Puzzles Creator (w_PuzzlesCreator)



Press **Update Save System (current scene)** (spot 1) and Press **Delete all save datas** (spot 2) to reset the save system.



Press **Play**.

Info: The text “**Activated**” is displayed in the console tab when the scene starts.

Save Extension (script needed):

This section allows to save and load your own parameters.

For example:

- Save if a door is open / closed.
- Save if a padlock is open or closed.
- Save the position of an object.

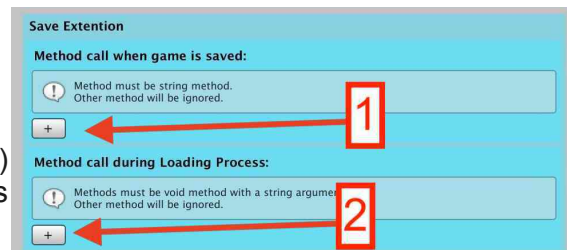
Save extension is divided in 2 parts:

Part 1: Methods call when game is saved (spot 1):

This section allows to save your own parameters.

Part 2: Methods call during loading Process (spot 2)

This section allows to load and use your parameters during the loading process.



How it works:

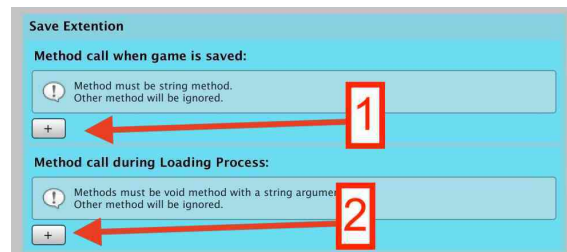
The section **Methods call when game is saved** is used to save your datas in a **string** method when the scene is saved (spot 1).

Important: Only a **string method** is allowed. Other methods types are ignored.

The section **Methods call during loading Process:**

- Is used when puzzles are initialized.
- Is used to load your datas in a **void** method with **one string argument** (spot 2).

Important: Only a **void method** with **one string argument** is allowed. Other methods types are ignored.



Save Extension Example (Tuto):

Example:

On a scene we have a door.

- We want to save if the door is opened or closed when the player saves the game.
- We want to load and initialized the door state during the scene loading process.

In the example we are going to save and load the state of 2 variables:

b_IsDoorUnlocked

b_IsDoorOpened

In this example, we are going to:

In your favorite script editor:

Step 1-Create a method to save the parameters

Step 2-Create a method to load and initialized the parameters.

In Unity:

Step 3-Setup **ObjIsActivated** script

Step 1: Create the string method to save the variables states:

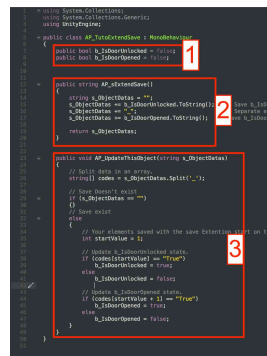
In the example we are going to save and load the state of 2 variables (spot 1):

b_IsDoorUnlocked
b_IsDoorOpened

Info:

To save the state of our variables a **string method** is needed (spot 2).

To load the state of our variables a **void method with a string argument** is needed (spot 3).



```
using UnityEngine;
using System.Collections;
using UnityEngine.SceneManagement;
using UnityEngine.UI;

public class AP_TutoExtendSave : MonoBehaviour
{
    public bool b_IsDoorUnlocked = false;
    public bool b_IsDoorOpened = false;

    public string AP_sExtendSave()
    {
        string s_ObjectData = "";
        s_ObjectData += b_IsDoorUnlocked.ToString();
        s_ObjectData += ",";
        s_ObjectData += b_IsDoorOpened.ToString();
        return s_ObjectData;
    }

    public void AP_LoadFromObject(string s_ObjectData)
    {
        // Split data in 2 arrays
        string[] s_data = s_ObjectData.Split(",");

        // Load Door's state
        if (s_data[0] == "True")
        {
            b_IsDoorUnlocked = true;
        }
        else
        {
            b_IsDoorUnlocked = false;
        }

        // Load Door's state
        if (s_data[1] == "True")
        {
            b_IsDoorOpened = true;
        }
        else
        {
            b_IsDoorOpened = false;
        }
    }
}
```

1: public bool b_IsDoorUnlocked = false;
2: public string AP_sExtendSave()
3: public void AP_LoadFromObject(string s_ObjectData)

To do that:

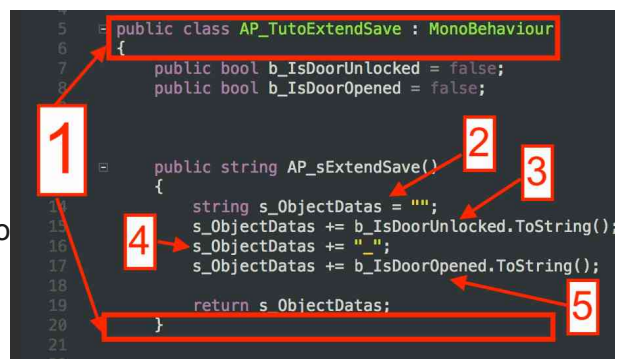
-Create a method that return a string (spot 1)

-Create a string named **s_ObjectData** (spot 2).

-Add to the string **s_ObjectData** the state of **b_IsDoorUnlocked** and convert this boolean value to a string (spot 3).

-Add **_** (underscore) to separate each element that need to be saved (spot 4).

-Add to the string **s_ObjectData** the state of **b_IsDoorOpened** and convert this boolean value to a string (spot 5).



```
5 public class AP_TutoExtendSave : MonoBehaviour
6 {
7     public bool b_IsDoorUnlocked = false;
8     public bool b_IsDoorOpened = false;
9
10    public string AP_sExtendSave()
11    {
12        string s_ObjectData = "";
13        s_ObjectData += b_IsDoorUnlocked.ToString();
14        s_ObjectData += ",";
15        s_ObjectData += b_IsDoorOpened.ToString();
16        return s_ObjectData;
17    }
18 }
19
20
21
```

1: public class AP_TutoExtendSave : MonoBehaviour
2: public string AP_sExtendSave()
3: s_ObjectData += b_IsDoorOpened.ToString();
4: s_ObjectData += ",";
5: return s_ObjectData;

Step 2: Create the void method with 1 string argument to load the variables states:

-Create a void method with one string argument (spot 1)

-Create a string array named codes to split datas in an array (spot 2).

-Create a if condition for the case where there is no existing save (spot 3a).

-Create the else condition to initialized the variables (spot 3b).

```
22 public void AP_UpdateThisObject(string s_ObjectDatas)
23 {
24     // Split data in an array.
25     string[] codes = s_ObjectDatas.Split(',');
26
27     // Save Doesn't exist
28     if (s_ObjectDatas == "")
29     {
30         // Save exist
31         else
32         {
33             // Your elements saved with the save extension start on the elements 1. Element 0 is used to save the object state.
34             int startValue = 1;
35             // Update b_IsDoorUnlocked state.
36             if (codes[startValue] == "true")
37             {
38                 b_IsDoorUnlocked = true;
39             }
40             else
41             {
42                 b_IsDoorUnlocked = false;
43             }
44             // Update b_IsDoorOpened state.
45             if (codes[startValue + 1] == "true")
46             {
47                 b_IsDoorOpened = true;
48             }
49             else
50             {
51                 b_IsDoorOpened = false;
52             }
53         }
54     }
55 }
```

Annotations in the image:

- 1: Points to the method signature `public void AP_UpdateThisObject(string s_ObjectDatas)`.
- 2: Points to the array declaration `string[] codes = s_ObjectDatas.Split(',');`.
- 3a: Points to the `if (s_ObjectDatas == "")` condition.
- 3b: Points to the `else` block.
- 4: Points to the `int startValue = 1;` line.
- 5: Points to the `if (codes[startValue] == "true")` condition.
- 6: Points to the `if (codes[startValue + 1] == "true")` condition.

important: Element 0 in the string array codes is used to save if the object is activated or deactivated in the scene.

The Elements saved in the Save Extension must start at element 1. It is the reason why the variable `startValue = 1` is created (spot 4)

Then update the `b_IsDoorUnlocked` state using the first value saved in the previous section (spot 5).

Then update the `b_IsDoorOpened` state using the second value saved in the previous section (spot 6).

Step 3: Unity Setup

-Create an empty object in your scene and rename it `MyObject` (spot 1).

-In the `w_PuzzleCreator_Pc` window tab (`Tools → Puzzles → Puzzles Creator`) select `02 ObjIsActivated` (spot 2).

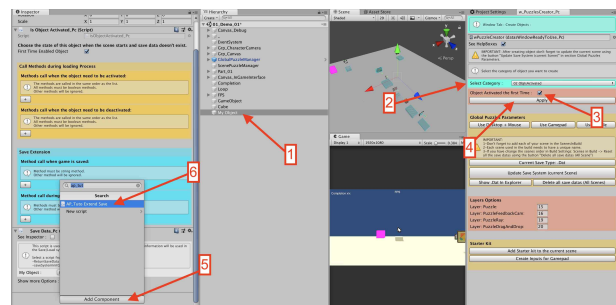
-Check the box `Object Activated the first time` (spot 3)

-Press `Apply` (spot 4)

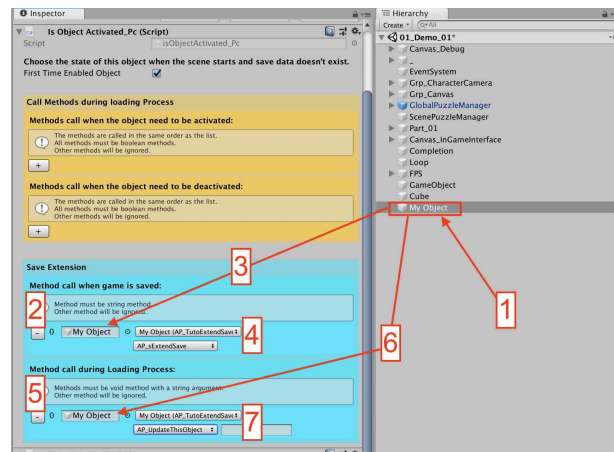
-In the Inspector press `Add Component` (spot 5)

-Choose `AP_TutoExtendSave.cs` (spot 6).

(This script contains the variables and the methods used for this example.)



- Select **MyObject** in the Hierarchy (spot 1)
- In the Inspector press the button **+** (spot 2)
- Drag and drop **MyObject** in the empty field (spot 3)
- Choose the script **AP_TutoExtendSave** and the string method **AP_sExtendSave** (spot 4).
- Press the button **+** (spot 5)
- Drag and drop **MyObject** in the empty field (spot 6)
- Choose the script **AP_TutoExtendSave** and the string method **AP_UpdateThisObject** (spot 7).



Info: Now **MyObject** is able to save and load the **b_IsDoorUnlocked** and **b_IsDoorOpened** parameters

Summary:

- 1-The elements are saved as a **string**.
- 2-Use a **string method** to save your elements.
- 3-Between each element to save add an underscore **_**
- 4-Initialize your elements in a **void method** with **one string argument**.

The Character.

The asset is designed to use your own character.

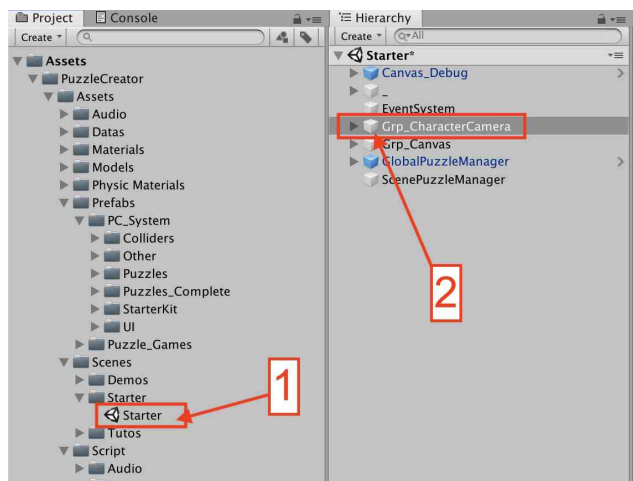
However you can use the character provided with the asset.

This section explains how to setup the character provided in the asset depending different scenario.

Case 1: Ready to use

You use the starting scene named **Starter** (spot 1): (*PuzzleCreator → Assets → Scenes → Starter → Starter*)

The character is already integrated by default (spot 2). (*Hierarchy: Grp_CharacterCamera*)

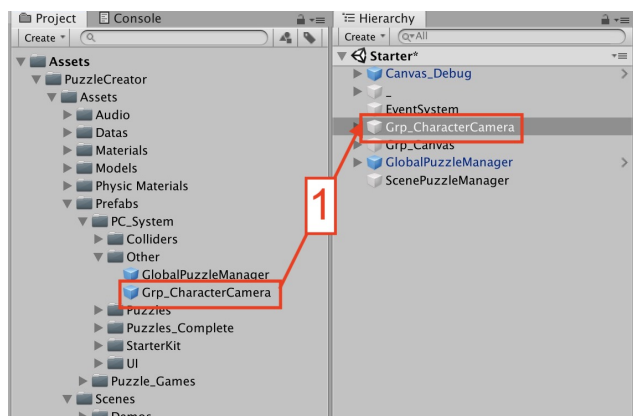


Case 2: Desktop

If you want to integrate the character to an existing desktop project:

Drag and drop the prefab

Grp_CharacterCamera in your scene (spot 1) (*PuzzleCreator → Assets → Prefabs → Pc_System → Other → Grp_CharacterCamera*)



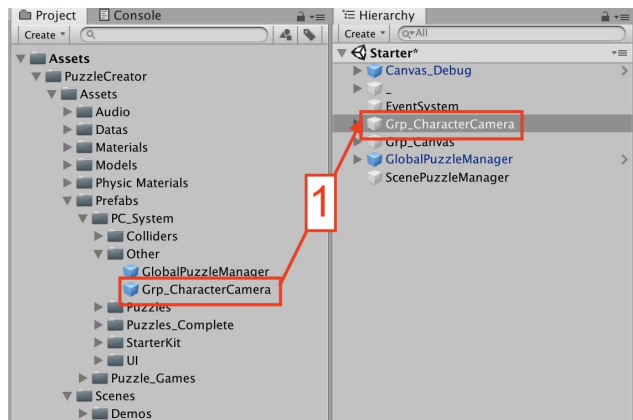
Case 3: Mobile

If you want to integrate the character to an existing mobile project:

Drag and drop the prefab

Grp_CharacterCamera in your scene (spot 1)

(PuzzleCreator → Assets → Prefabs → Pc_System → Other → Grp_CharacterCamera)

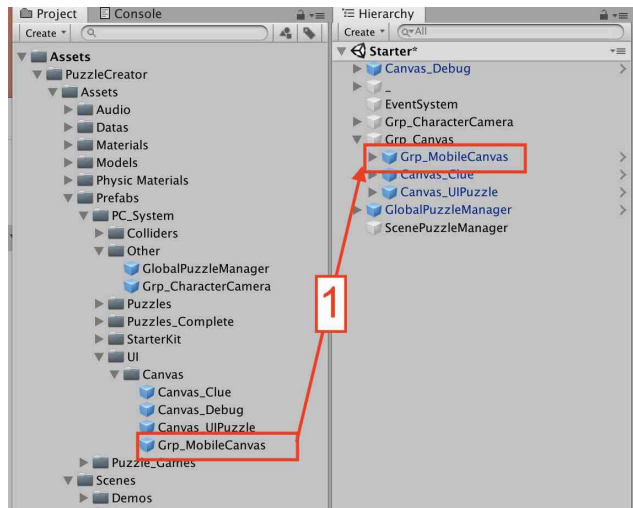


Drag and drop the prefab **Grp_MobileCanvas** in your scene (spot 1)

(PuzzleCreator → Assets → Prefabs → Pc_System → UI → Canvas → Grp_MobileCanvas)

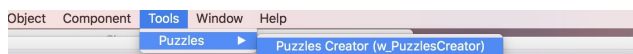
Info: The **Grp_MobileCanvas** prefab only works with the character provided in the asset.

If you want to use the UI with your character you will have to code the connection between UI and character by yourself.



Open the window **w_PuzzlesCreator**:

(Tools → Puzzles → Puzzles Creator (w_PuzzlesCreator))

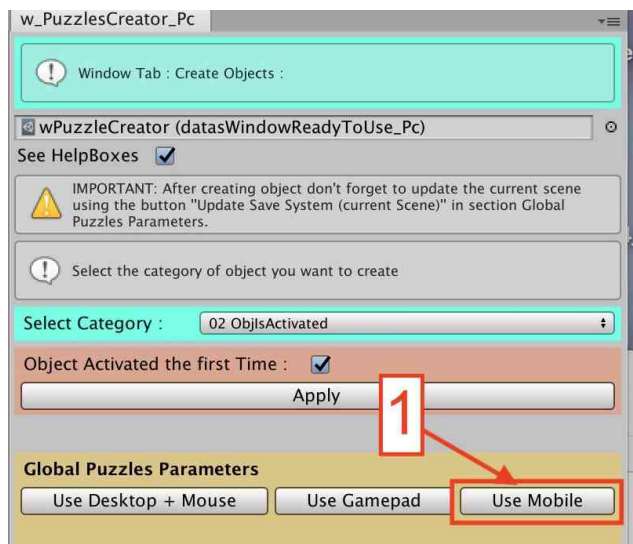


In window **w_PuzzlesCreator** press button **Use Mobile** (spot 1)

Info:

This button allows to:

-Activated the Mobile UI inputs for the character if the prefab **Grp_MobileCanvas** is in the Hierarchy.



Debugger

This object help you debug puzzles more easily:

How to use the debugger:

-Drag and drop into your scene the prefab

Canvas_Debug (spot 1)

Project tab: Assets → PuzzleCreator → Assets → Prefabs → PC_System → UI → Canvas → Canvas_Debug

By default when scene is in Play Mode:

-Press **F** to finish a puzzle without solving it.
-Press **F** a second time to cancel the debugger action.

-Press **G** to bypass conditions to open a puzzle.
-Press **G** a second time to cancel the debugger action.

Customize debugger parameters:

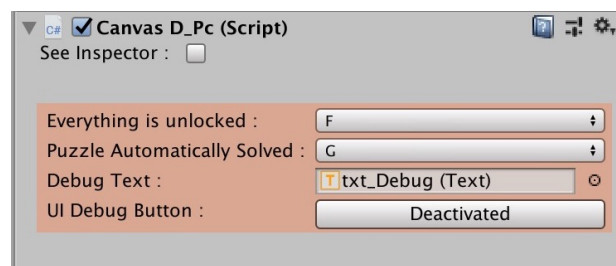
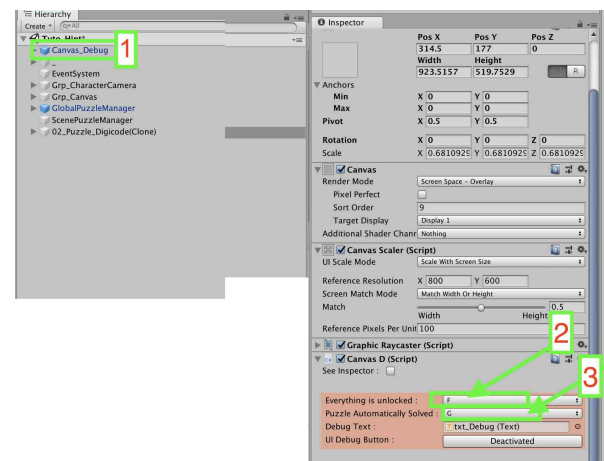
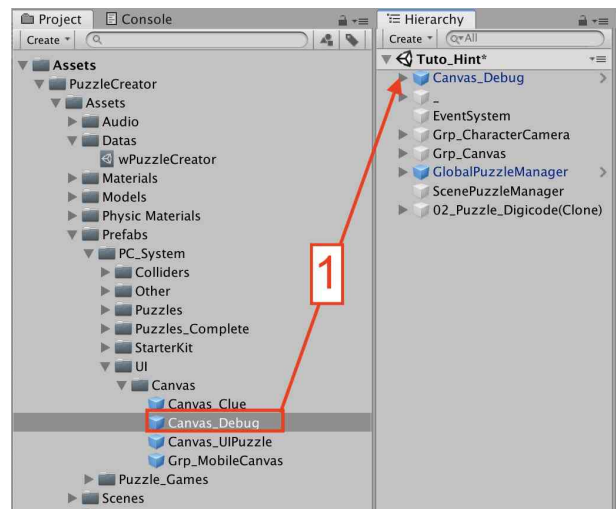
-Select **Canvas_Debug** in the **Hierarchy** (spot 1).

-Select the input in the dropdown list to bypass conditions to open a puzzle (spot 2).

-Select the input in the dropdown list to finish a puzzle without solving it (spot 3).

Customize debugger for Mobile:

-Click on the button next to UI Debug Button to activate or deactivate the UI Debug button.

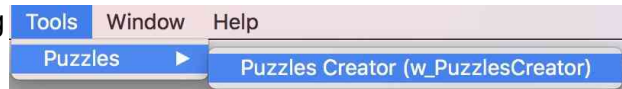


Troubleshooting

- 1-The scene doesn't work / There is an error in the console [link](#)
- 2-Current scene save is not erased after pressing update Save System (current scene) [link](#)
- 3-Puzzle Detection issues [link](#)
- 4-The character controls are reversed [link](#)

1-The scene doesn't work / There is an error in the console tab

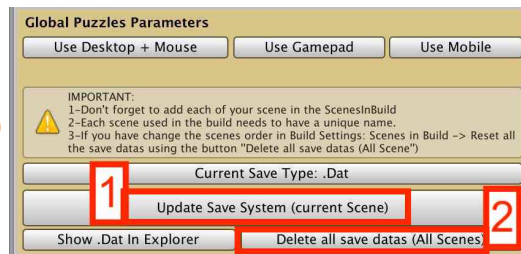
If you have an error in the console tab the first thing to try is:



-Go to: **Tools** → **Puzzles** → **Puzzles creator (w_PuzzlesCreator)**

-Press button **Update Save System (current scene)** (spot 1)

-Press button **Delete all the save datas (All scenes)** (spot 2)



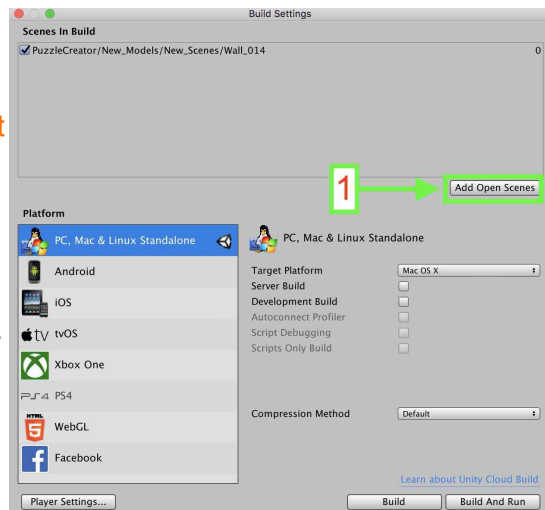
2-Current scene save is not erased after pressing update Save System (current scene)

When the button **Update Save System (current scene)** is pressed save datas for this scene are deleted only if the scene is part of build scenes.

-Go to **File** → **Build Settings**
-Press **add Open Scene**

Now when the button **Update Save System (current scene)** is pressed the puzzle datas are deleted for this scene.

Important: If you change the scene order in the **SceneInBuild** window you need to press **Delete all save datas (All Scenes)** to update the save system.



3-Puzzle Detection issues

If 2 puzzles are too close to each other it can cause issues.

(More info in the sub-section **Detection Restrictions** section **Puzzle Detection Type in Doc Part 2**)

4-The character controls are reversed

If the character controls are inverted when you use the keyboard:

-Go to: Tools → Puzzles →
Puzzles creator (w_PuzzlesCreator)

-In section Global Puzzles Parameters press button
Use Keyboard + Mouse (spot 1)



Scripting:

Scripting: How to create a boolean method

- On project tab click on **Create** button (spot 1)
- On the dropdown menu choose **C# script** (spot 2)

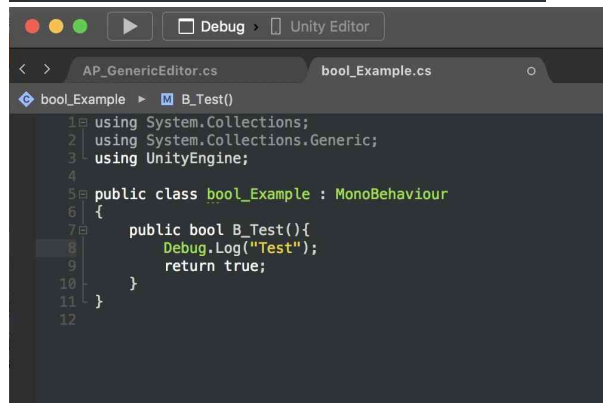
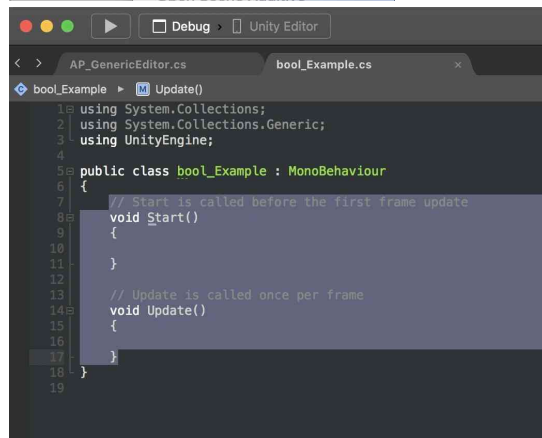
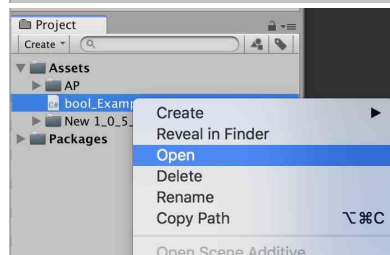
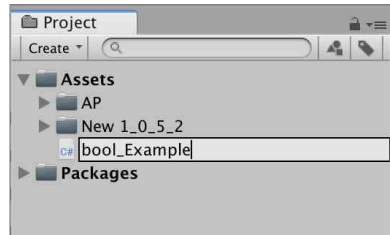
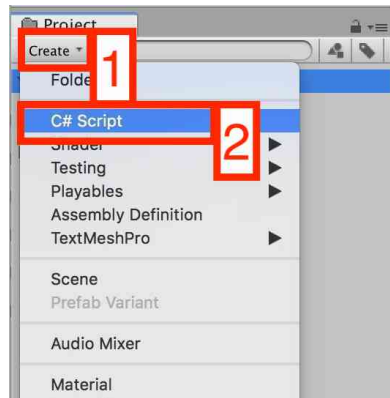
Rename the script **bool_Example**

- Right click on script **bool_Example**.
- On the dropdown menu choose **Open**.

-Select and delete line **7** to **17** (method start and Update).

Write this:

```
public bool B_Test(){  
    Debug.Log("Test");  
    return true;  
}
```



How it works:

public (spot 1):

Allows to access this method outside this script.

bool (spot 2):

Define that it is a boolean method.

B_Test (spot 3):

It is the name of the method.

Debug.Log (spot 4):

In this example we write **Test** in the console when the method is call. Write your code here.

Return true; (spot 5):

A boolean method return True or False

In our case the method must return True at the end of the method.

This allows the modules contained in the asset to know that the method is complete.

In the asset it is possible to call a list of boolean methods. These methods are called one after the other. When the first method is finished its returns True. With that information it is possible to call the following method and so on.

Scripting: Puzzle Inputs:

Switch between inputs types at runtime:

From any script you can switch inputs to:

Desktop input with:

```
AP_GlobalPuzzleManager.instance.AP_SwitchPuzzleInputsToKeyboardAndMouse(true);
```

Gamepad input with:

```
AP_GlobalPuzzleManager.instance.AP_SwitchPuzzleInputsToGamepad(true);
```

Mobile input with:

```
AP_GlobalPuzzleManager.instance.AP_SwitchPuzzleInputsToMobile();
```

Access inputs at runtimes:

Keyboard Inputs:

Validation button (KeyCode Type):

```
AP_GlobalPuzzleManager.instance.validationButtonKeyboard
```

Exit Puzzle (Mode Focus) (KeyCode Type):

```
AP_GlobalPuzzleManager.instance.backButtonKeyboard
```

Gamepad Inputs:

Cursor Horizontal (Mode Focus) (string Type):

```
AP_GlobalPuzzleManager.instance.HorizontalAxisJoystickLeft
```

Cursor Vertical (Mode Focus) (string Type):

```
AP_GlobalPuzzleManager.instance.VerticalAxisJoystickLeft
```

Validation button (KeyCode Type):

```
AP_GlobalPuzzleManager.instance.validationButtonJoystick
```

Exit Puzzle (Mode Focus) (KeyCode Type):

```
AP_GlobalPuzzleManager.instance.backButtonJoystick
```

Scripting: Useful Methods or variables for AR:

By default a puzzle is activated and available:
If the player click on the puzzle
or enter the trigger PuzzleDetector.

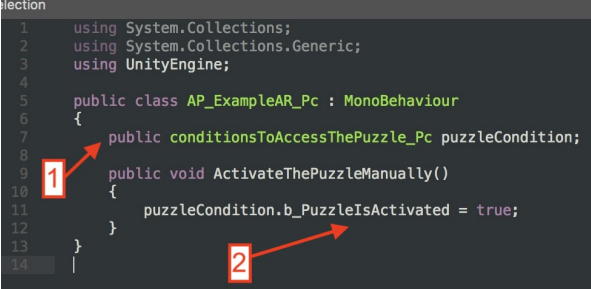
In AR it is not possible. So it needed to activate manually a puzzle.

How to do that:

Change the state of `b_PuzzleIsActivated` in the script `ConditionsToAccessThePuzzle_Pc.cs` that can be find in the puzzle Inspector.

-Declare a variable to access the script `ConditionsToAccessThePuzzle_Pc` (spot 1).

-Call a method to change to `true` the state of the variable `b_PuzzleIsActivated`;



```
1 using System.Collections;
2 using System.Collections.Generic;
3 using UnityEngine;
4
5 public class AP_ExampleAR_Pc : MonoBehaviour
6 {
7     public ConditionsToAccessThePuzzle_Pc puzzleCondition;
8
9     1 public void ActivateThePuzzleManually()
10     {
11         puzzleCondition.b_PuzzleIsActivated = true;
12     }
13 }
14
```

How to add a method (overview):

Even though the sections are different, the process for adding methods remains the same:

In several sections of the puzzle system it is possible to add custom methods.

Step 1

-Add your script to an object of the Hierarchy. (Spot 1)

Step 2

-In Inspector go to the section where you want to add your method (spot 2)

Step 3

-Create a new slot by pressing + (spot 3)

Step 4

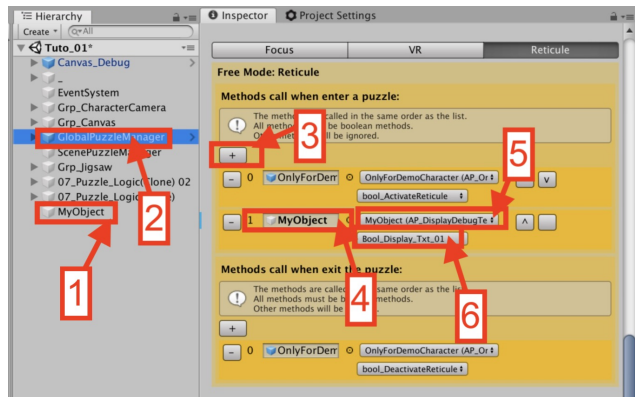
-In the Inspector, **drag and drop** the object of the Hierarchy including your script into the **new empty slot** (spot 4).

Step 5

-Choose in the dropdown menu the script that contains the method you want to use (spot 5).

Step 6

-Choose in the second dropdown menu your **method** (spot 6).



Scripting: Check if the access to a puzzle is allowed or denied

We will take an example:

There is a key and a puzzle in the scene.
Access to the puzzle is only possible if the key is present in the player's inventory.

For optimization reasons:

We don't want that the puzzle check each frame that the key is in the player's inventory.

What to do?

Once the key is in the player's inventory:
You have to ask the puzzle to check if the conditions of access to the puzzles are done
(In our case: The key must be in the player's inventory).

How to do it?

In each puzzle you have a `conditionToAccessPuzzle.cs` script.
In this script is the `checkAccessAllowed()` method checks if the conditions are met to access this puzzle.

To call this method you must:

Access the `conditionToAccessPuzzle.cs` script of your puzzle (spot 1).
Call the `checkAccessAllowed()` method (spot 2).

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class AP_UpdatePuzzleAccess : MonoBehaviour
{
    public conditionsToAccessThePuzzle_Pc_conditions; // Reference to the puzzle you want to check its access
    public GameObject ObjectToActivate; // Reference to the object we want to activate in the Hierarchy

    void Update()
    {
        if (Input.GetKeyDown(KeyCode.Y)) // When key Y is pressed
        {
            if (ObjectToActivate.GetComponent<Renderer>().enabled == false)
            {
                ObjectToActivate.GetComponent<Renderer>().enabled = true;
            }
            else
            {
                ObjectToActivate.GetComponent<Renderer>().enabled = false;
            }
            conditionToAccessThePuzzle_Pc_conditions.checkAccessAllowed(); // Activate in the hierarchy ObjectToActivate
            // Check if the selected puzzle access can be allowed
        }
    }
}
```

To have another example read:

[Doc 1](#) section [Tuto 1](#)
sub-section [Add a condition to start a puzzle.](#)

