InstantGUI
Main Menu
Continue
Start New
Settings
Credits
Quit
Settings
Game
Video
Audio
Difficulty:
Normal
Player Name:
Password:
Tutor
Easy
Normal
Hard
Nightmare
Invert Mouse Y
Mouse Speed
Return
Credits

InstantGUI is a GUI display system and GUI editor. It allows to create full featured interface system for the game with maximum convenience and as fast as prototyping it!

GUI creation is done with no scripting. Now an artist or game designer can create interface the way they like. Of course there a script interactions with GUI which are done with events

Starting GUI creation

All the GUI creation and editing is done in Game view in WYSYWIG. The interface you create will look exactly as you see it while editing.

All user interface consists of Elements – small interface portions with individual functions: button, checkbox, window, etc. Unlike scripted Unity GUI, all elements of InstantGUI are separate game objects, and that gives the ability to quickly find, sort, and edit elements using Hierarchy and Game view. Elements form a hierarchy structure: if an element contains other elements (for example like window contains buttons) all the containing elements should it's be children. All of the elements are parented to root object with InstantGui script (named InstantGui as default).

To start creating GUI the root element should be created. Select GameObject → Create InstantGUI → New InstantGUI from a menu bar. A game object with “InstantGUI” name will be created.

Then the default interface Style Set should be assigned. Style Set defines interface visual appearance: textures, colors, fonts, etc. Style sets are saved as an asset files. Drag “Glow.asset” from Demo folder to the root element's Style/Style Set slot. This will make all the new elements use Glow style set as the default.

The next step is to create the element of your choice – for example main window: Select GameObject → Create InstantGUI → Window from a menu bar.

Switch to the Game view and select the element that was just created in a hierarchy view. An editor frame with controls will appear around element. Dragging rectangular controls (anchors) attached to the sides of the element change it's relative position, and circle controls at the corners change element's pixel offset.

Element uses relative (in percent) and per-pixel positioning simultaneously. Firstly element is placed using relative position in a space of parent element, and then pixel offset is applied. For example if left and right relative anchors set to left side of a parent, the element will be bound to the left side of a parent and element will have a constant size. If left anchor is at the left side of a parent, and the right anchor is on the right the element's size will be shrinking and expanding with the size change of a parent. If both anchors set to 50% between left and right element with a constant size will be fixed to the center of parent element. The pixel offset will offset the element after it was placed relatively.

Dragging relative or offset controls with **Control** key pressed will move these controls with a grid snap.
Dragging with **Shift** pressed will move control in one dimension only.
You can move element changing it's offset by clicking inside it's rectangle and dragging it to the desired place.
Using **arrow keys** with selected element will move it in relevant direction.
Clicking on another element will select it.
Alt-clicking on element will activate it. For example alt-clicking on button will press it the way it was pressed in game. Please note that it works only with element activation, other actions such as typing text in InputText or scrolling Slider will work only in playmode.
Press **L** to lock selection. When selection is locked no element could be selected by clicking it, but you can select element from hierarchy view. Press **H** to toggle editor controls on or off.

Common Element Settings

Each element has two visual components – text label and texture. Internally, it has one or more GUIText and GUITexture components to set text and texture respectively. Text is set with the "Text" control and it is always unique for each element, while texture is set with style.

Text sets the displayed label of element. If **Use Object Name** is turned on then text will match the object name and vice versa – if the text has at least one char length. Turn Use Object Name off if you want to set text from script.

Script properties:

Property:	Type:	Description:
text	String	
guiLinkText	boolean	Internal name of Use Object Name. Turn this off before setting text with script

Style

Styles set visual appearance of the element – textures, fonts, effects, etc. Styles are grouped in a style set assets, which could be saved as .asset files. To apply a style to element style set should be assigned in Style Set slot, and specific style should be chosen from the style set with Style popup.

Some elements can have their own unique styles, for example some specific buttons, glyphs, icons, etc. It is not convenient to clutter style set with such styles, that's why element can have it's own Custom Style. Custom style settings have the standard style interface which is described in the relevant section.

Script properties:

Property:	Type:	Description:
styleSet	InstantGuiStyleSet	
customStyle	boolean	
styleName	String	To find style in style set
style	InstantGuiStyle	

Position

Elements are structured like standard Unity objects – they form a hierarchy that could be changed with hierarchy view. Every element inherits it's parent position, and the change of element's position or size causes the change in it's children. The hierarchy root element should have InstantGui script attached.

All the position adjustments are done with the element's Position control group. Transform component has almost no effect on proper element positioning, it should not be changed. All the needed data in transform (like position z) is set with InstantGuiElement component. Transform position and scale x and y and should be set to zero.

Element uses relative (in percent) and per-pixel positioning simultaneously. Firstly element is placed using **Relative** position in a space of parent element. For example position of 0,100,0,100 (left, right, top, bottom) will place element the same as parent, and 50,50,50,50 will place element in the center of it's parent, but it's size will zero.

After the element was positioned relatively, **Offset** is applied to it's position. Offset is measured in pixels and shifts element from it's relative position. For example, with offset the above-mentioned zero-sized center element can get it's dimensions: 30,-30,10,-10 offset will set the size to 60x20.

Final **Absolute** position in pixels is displayed information purposes.

Zero coordinates are located in top left corner.

Setting one of the values in **Preset** popup will quickly set Relative and Offset positions to the selected stock parameters.

Layer Offset changes z-position of an element relative to it's parent. Element with bigger z-position displayed on top of lesser z-positioned elements. The default layer offset is 1, which makes every child element to be displayed on top of it's parent with z-position +1. To place element always on top or on background set layer offset to large positive or negative values.

Lock Position freezes element preventing it's position from change with editor in game view. However, you can input values in the inspector.

Each style has a default element relative and offset values. To use style default instead of unique element positioning turn on **Use Style Placement**. This can be useful for placing auxiliary elements like window close buttons which has the same style and same position relative to their parent elements.

Set as Default in Style button saves current position as the default placement in a current style. All newly created elements of this type and elements with "Use Style Placement" turned on will use this relative, offset and layer offset.

Script properties:

Property:	Type:	Description:
useStylePlacement	boolean	
relative	InstantGuiElementPos	
offset	InstantGuiElementPos	
absolute	InstantGuiElementPos	It is not recommended to set absolute value directly
layerOffset	int	
lockPosition	boolean	

Attributes

- **Dynamic**: only dynamic elements involved in the mouse pointing calculations. If element is not dynamic, it cannot be clicked in game and in editor. Turn "dynamic" off for the static labels that cannot be activated by mouse.
- **Editable**: editable element can be selected by clicking in editor. False "editable" state is used for the elements that commonly are autmaticaly-adjusted – like slider thumb or list item. However non-editable elements could be selected from hierarchy.
- **Pointed**: objects are pointed when mouse cursor is inside element active rectangle and the element lies at the top layer in this mouse cursor position. In other words element is pointed when mouse is "hovering" above it.
- **Pointed On Background**: objects are pointed on background when mouse cursor is inside element active rectangle, no matter if this element is on top or not.
- **Disabled**: the element could not be activated and displays a disabled style (if any). Use this for controls that should be displayed, but user has no access to them.
- **Activated**: becomes true for the element **for one frame only** after user has activated (pressed and released) it. This can be used for determining button pressing in your scripts: if (buttonElement.activated) {//do some action}. For instant activation (to activate right after element was pressed) use "instant" attribute. Element could be activated in editor with Alt-click.
- **Pressed**: element is pressed when user is holding mouse pressed over the element. In contrast to "activated", element is pressed as long as user holding it.
- **Checked**: checked state is used in such elements like toggles, checkbuttons and tabs. When element is in "checked" state activated style is displayed (if any).
- **Instant**: element is activated right after pressing. Otherwise it is activated on mouse up after holding pressed.
- **Password**: element text is displayed in dots. This is useful for password input. Please note that internal element text remain, so password attribute fit only for visual hiding password, and cannot give a reliable security.

Script properties:

Property:	Type:	Description:
dynamic	boolean	
editable	boolean	
pointed	boolean	
pointedOnBackground	boolean	
pointedTime	float	How much time this element is under cursor. Increment on pointed, resets when element becomes not pointed
unpointedTime	float	How much time this element is not under cursor
activated	boolean	
pressed	boolean	
check	boolean	Named “Checked” in Inspector
instant	boolean	
password	boolean	

Activators

Most of the elements can perform some action on a typical usage. These actions are specified in Inspector in form of “On *Action*” menu, that sets the parameters of an activator class. For example button has an On Pressed activator.

Each activator can enable and disable objects that are set in Enable Objects and Disable Objects arrays. Usually these objects are other elements: for example pressing game's “Settings” menu button a settings menu should appear by enabling it with an activator. In this case settings menu game object should be disabled by-default, and it should be in button activator's Enable Objects list. Clicking “Cancel” button in settings menu will close the settings menu if it is in button's Disable Objects list. To add game object to the list drag-and-drop it from Hierarchy view to the corresponding object field.

Moreover, activators can send messages to other game object calling SendMessage() function. This message will let your game scripts know that user activated specific element. To turn sending message on just type the name of a function that receives a message and Message Direction. Try sending message to All game objects as seldom as possible due to performance reasons.

Element variable will be sent along with a message to make your script know which element caused a message. So message receiving function should look like Receive`Message(element:InstantGuiElement)`.

Script properties:

Property:	Type:	Description:
enableObjects	GameObject[]	
disableObjects	GameObject[]	
message	String	
messageRecievers	enum MessageRecievers	{none, current, upwards, broadcast, gameObject, all}
messageGameObject	GameObject	

Button
Element for all sorts of standard buttons, including window close button, slider increase and decrease buttons, etc. Button element is quite simple. It has only one activator that activates on button pressing.

Please note that checkbutton and radiobuttons do not belong to buttons, actually, they are toggles.

Script properties:

Property:	Type:	Description:
onPressed	InstantGuiActivator	Activates when button is activated

Window
Background element for other controls. Window can be moved with all containing elements if the **Movable** attribute is turned on. The **Scape** parameter defines borders where window can be moved to:

- off: window movement is not limited
- screen: window can be moved withing game screen, and cannot cross it's borders
- parent: window can be moved within parent element's borders

Window has a **Close Button** element. Activating Close Button will close the window.

Script properties:

Property:	Type:	Description:
movable	boolean	
scape	InstantGuiWindowScape	Enum, {off, screen, parent}
closeButton	InstantGuiElement	If closeButton is activated window sets non-active

Toggle
Element that could be turned on or off: checkboxes, checkbuttons, tabs, etc. It has two activators: **On Checked** and on **On Unchecked**.

To make radio buttons there are two properties: **Uncheck Toggles** is an array of elements that will be un-checked after toggle is checked, and **Could Be Unchecked** boolean that allows toggle to be un-checked. Classic radio button should have Uncheck Toggles array filled with other buttons, and Could Be Unchecked set to false.

Script properties:

Property:	Type:	Description:
couldBeUnchecked	boolean	
uncheckToggles	InstantGuiElement[]	
onChecked	InstantGuiActivator	
onUnchecked	InstantGuiActivator	

Input Text
Element with text that could be changed in-game.

By default text could be edited only when text has Checked attribute set to true, but if Always Typing is set to true text is edited all the time text game object is active. Cursor Width and Cursor Color are the width and color of text cursor respectively. Cursor Texture is the text cursor symbol, cursor will not be displayed without a texture.

To make input text displayed as password use Password attribute.

Script properties:

Property:	Type:	Description:
isPassword	boolean	
alwaysTyping	InstantGuiElement[]	
cursorPos	int	Number of char at which cursor is which will be edited.
cursor	GUITexture	Cursor symbol GUI texture
cursorWidth	int	
cursorColor	Color	
cursorTexture	Texture	

Text Area

Element to display multi-lined scrollable text. This element is not intended for text editing, it just displays pre-set text.

Unlike Input Text text is stored not int Text variable, but in Raw Text. Text variable stores final formatted piece of text that is displayed on screen. Width and Height Adjust modifies width and height of displayed text to make margins. The text will shrink or grow from right and from bottom, so use Text Offset to place the text to the desired position. First Line is number of first line that is displayed in scrollable text.

It is not recommended to use Text Area for displaying single-line labels due to performance reasons, use simple Element instead.

To scroll a Text Area Mouse ScrollWheel Input axis should be set up in Unity Input Manager, otherwise an error will occur. This axis is set in any new project by default, if it does not add “Mouse ScrollWheel” axis in Edit->Project Settings->Input, or remove

```
        slider.value -= Input.GetAxisRaw("Mouse ScrollWheel")*10;
```

line in TextArea script, or set another axis in that line.

Script properties:

Property:	Type:	Description:
rawText	String	Raw unformatted text
firstLine	int	
widthAdjust	int	
heightAdjust	int	
slider	InstantGuiElement	

Slider

Could be in horizontal and vertical variant. Variants are switching by Horizontal toggle.

Value sets current slider thumb position. Min and Max sets slider limits: minimum and maximum values respectively. And shown value determines the size of a thumb in values. For example, if you want to scroll 20-lines text with 8 lines displayed, Shown Value will be 8 and Max will be 12 (20-8).

If Step is not equal to zero value is rounded to Step on each change. On every Increase Button and Decrease Button activation value is changed on ButtonStep. Slider Diamond is an element of slider thumb.

Script properties:

Property:	Type:	Description:
value	float	
min	float	
max	float	
step	float	
buttonStep	float	
shownValue	float	
diamond	InstantGuiElement	
incrementButton	InstantGuiElement	
decrementButton	InstantGuiElement	
horizontal	boolean	

List

Displays a scrollable list of selectable values. Useful for quality chooser, game level selector, players list, etc. Values are displayed as strings stored in Labels array. To Add new label press Add button. Each of the displayed values is a separate element.

Selected attribute corresponds to the value currently selected.
Line Height sets the height of each value element line.
A List could be scrolled by changing First Shown value. It defines the number of element that is placed on the first line.
Slider Margin is the elements' offset from the right. It is made to place slider within List borders.
Element Style is a style used by line elements.

Script properties:

Property:	Type:	Description:
labels	String[]	
elements	InstantGuiElement[]	
lineHeight	int	
firstShown	int	
selected	int	
slider	InstantGuiSlider	
sliderMargin	int	
elementStyleName	String	To change style after it was selected in editor popup
elementStyle	InstantGuiStyle	

Popup

Displays a list of selectable values on click.

Script properties:

Property:	Type:	Description:
list	InstantGuiList	

Tabs

A field with a switchable content. The content is switched with tabs elements. A special container element – Field – corresponds to each Tab element. To create new Tab drag-and-drop it to the object field below “Tabs:” label. A Field to the new tab will be created autocratically, but it can be replaced by dragging new element to the corresponding object field.
Selected attribute corresponds to the tab number currently chosen.

Script properties:

Property:	Type:	Description:
tabs	InstantGuiElement[]	
fields	InstantGuiElement[]	
selected	int	

Style and Style Set

The visual representation of elements is done with the Style. Commonly styles are kept in a Style Set asset file. To set a style to element assign style set asset file and select one of the styles in style set.

However, an element can have it's own custom style. It is not convenient to save icons, symbols or other unique element to the style set, so you can turn on the “**Custom Style Set**” toggle and assign style directly in element.

Styles could be edited in inspector when the asset file or element with a custom style is selected.

Every style should have a unique **name**. Styles without name could not be assigned. Be careful renaming styles in style set – each element gets a style using it's name, so renaming the style requires it's re-assigning in elements.

Every style has four sub-styles. Each of the sub-styles determines the appearance of the element in different states:

- **Default**: element's common visual appearance, when it is not active, hovered by mouse pointer or disabled.
- **Pointed**: element's appearance is blending to this state (using Blend Speed parameter) when the element is under mouse pointer.
- **Active**: element is activated or checked
- **Disabled**: element is in the disable state.

Each sub-style can be turned on or off. If the state should not be used (for example the button could not be disabled) it's sub-style should be turned off. Please note that **if none of sub-styles is turned on element will not be displayed**.

Every sub-style can have it's own **Texture**, **Text Color** and **Text Offset**. Sub-style text offset is added (summed) with the whole style's text offset. For example if the pressed button's text should be 2 pixels lower than it's default text active state's text offset should be 0,-2.

Texture borders sets the pixel inset of element texture's borders. It sets the number of pixels that are not affected by element size. This is similar to Unity GUITexture borders. Pressing **Half Size** will set all the borders to the half of texture's width and height.

Fixed Width and **Fixed Height** will turn off the texture stretching and make the element's size constant. Fixed size enables by the corresponding toggle, and the size itself is set in int field. Pressing Get button will set the size to the texture size.

Proportional will set element's visual width or height using the given aspect ratio.

Click Offset determines the borders from the sides of the element that are ignored by mouse clicking. This could be useful for making element's glow or shadows. With Click Offset the active area of an element could be reduces compared to the element's texture.

Parameters that sets element's text appearance are: **Font, Font Size, Text Alignment** and **Text Offset**. Text alignment is style preset for placing text in an element. Text offset is a general whole-style offset of the text.

Default Placement is the element's default relative, offset and layer offset positions. Each newly created element uses the default style position. Moreover, default position is applied when element' toggle “Use Style Position” is turned on.

Add To Style Set button (available for elements with custom styles) saves custom style to the assigned style set. Please note that pressing this button does not turn custom style off, so saved style will not be applied. To use the saved style turn “Use Custom Style” toggle manually.

Script interaction

In order to make a full-functional game project scripts should interact with InstantGui. This can be done in two ways: activator messages and read/write script variables directly.

Activator message is the analog of event. Most elements have activators, that are triggered on common element actions: on pressing button, or on changing slider value, etc. These activators could be configured in “On <Action>” foldouts in Inspector. To make an activator send message the message name should be entered in Message field. Element variable will be sent along with a message to make your script know which element caused a message. So message receiving function should look like ReceiveMessage(element:InstantGuiElement).

Sending messages is easy and universal approach – it does not matter what is your scripting language or what internal parameters do elements have. All is needed to use activators – is a just function with desired name and one InstantGuiElement as parameter. But this approach is not flexible – you cannot get all the element parameters, and you cannot set any of them.

The other approach is to directly get and set the element parameters. For example – do some action on button press:

```
//use this script on button object
using UnityEngine;

public class Test : MonoBehaviour
{
    public InstantGuiButton testButton;
    void Test()
   {
         if (testButton==null) testButton = GetComponent(InstantGuiButton);
         if (testButton.activated)
         {
                //do some action
         }
    }
}
```

Example of setting gauge value to hero health (presumed that you have Hero class with health and maxHealth parameters):

```
//use this script on gauge object
using UnityEngine;

public class SetGauge : MonoBehaviour
{
    public InstantGuiGauge healthBar;
    public hero : Hero;
    void Test()
   {
         if (healthBar==null) healthBar = GetComponent(InstantGuiGauge);
         healthBar.max = hero.maxHealth;
         healthBar.value = hero.health;
    }
 }
```

It will be more convenient to set the health gauge value in Hero script directly, thought. This script just illuminates how it could be done.