Jarcas Popup UI

v1.0

Instruction Manual

Introduction

Thank you for purchasing the Jarcas Popup UI (JPUI) system! JPUI is built on top of the excellent new Unity UI (uGUI) system in Unity 4.6. No third-party packages or plugins are required.

We designed JPUI in the hopes that it will make the task of both displaying popup dialogs and retrieving user input from them a simple and painless one. With JPUI you can display both modal popup menus that are centered on the screen as well as context popup menus that are centered over some other object on-screen.

Example Scene

We have included an example scene (*JarcasPopupExample*) showing how JPUI could work in-game. Each of the buttons in the scene will make a different call to show a popup dialog. The status text will also be updated with user input on calls that provide a UnityAction callback function.

JarcasPopupExample.cs contains all the example code used to display the various popups. This script contains examples of most usages of JPUI as described in the *API Reference* below.

Quick Start

If you want to quickly see how JPUI would function in your game, try placing an instance of the PopupManager prefab (in the *Example/Prefabs* folder) on your Unity UI Canvas. Make

sure that the PopupManager is the **last** UI GameObject in your Canvas hierarchy so that it will always display on top of any other UI that may be active.

The prefab comes with some sample popup panels built-in. Once it has been placed in the scene, you can immediately start making calls to the PopupManager from your scripts. (*see API Reference below*)

Integrating with your UI

If you want to use JPUI with your own panels, sprites, fonts, etc., then you'll need the following:

1. a Popup panel GameObject and/or a Context Popup panel GameObject
2. (optional) Animators for your Popup and Context Popup panels (see *Animated Popup Panels* below)
3. a *PopupManager* component placed somewhere in your scene

The *PopupManager* component must then be given the following references to your various UI panels and widgets. Note that some references may be left blank if you do not intend to use those features.

- ***popupPanel*** - The popup panel GameObject that will be activated when popups are shown
- ***popupButtonParent*** - The RectTransform in the popup panel that buttons will be parented to when they are instantiated
- ***popupButtonPrefab*** - The prefab that will be instantiated when buttons are added to the popup panel
- ***textWidget*** - The Text component that holds the main text of the popup panel
- ***inputWidget*** - The InputField component that will be used for input popups
- ***textHighlightColor*** - A highlight text color that can be universally adjusted across all popup text. Highlight your text by using the <color=highlight> tag
- ***contextPopupPanel*** - The context popup panel GameObject that will be activated when context popups are shown
- ***contextButtonParent*** - The RectTransform in the context popup panel that buttons will be parented to when they are instantiated
- ***contextButtonPrefab*** - The prefab that will be instantiated when buttons are added to the context popup panel

Animated Popup Panels

JPUI can just simply activate and deactivate your popup panels, but to really spice up your popups JPUI supports Mecanim animations for opening and closing the popup panels. Here are the requirements for animated popup panels:

- Animator component on the popup panel GameObject

- Animator Controller must have:
    - a bool param named “open” that will trigger the animation to open the panel
    - a state named “Closed” that signifies the panel is finished closing

Examine the setup of the example popup panels for further reference.

API Reference

JPUI is contained within its own namespace, *Jarcas*, so before making any calls you will need to add a using statement to the top of your script:

```
using Jarcas;
```

All JPUI calls are then made to the PopupManager singleton via this syntax:

```
PopupManager.instance.MethodName()
```

Public JPUI methods

ShowPopup`( string text )`
- Shows the popup w/ an OK button and no callback

ShowPopup`( string text, UnityAction< int > callbackMethod )`
- Shows the popup w/ an OK button
- UnityAction callback will be called when OK is pressed

ShowPopup`( string text, string buttonText, UnityAction< int > callbackMethod )`
- Shows the popup w/ one button w/ custom text
- UnityAction callback will be called when button is pressed

ShowPopup`( string text, string button1Text, string button2Text, UnityAction< int > callbackMethod )`
- Shows the popup w/ two buttons w/ custom text
- UnityAction callback will be called when button is pressed
- int callback parameter will contain the 0-based index of the button that was pressed

ShowPopup`( string text, string button1Text, string button2Text, string button3Text, UnityAction< int > callbackMethod )`
- Shows the popup w/ three buttons w/ custom text
- UnityAction callback will be called when button is pressed
- int callback parameter will contain the 0-based index of the button that was pressed

ShowLoadingScreen`( string loadingText )`

- Shows a popup with no buttons (usually used as a loading screen). Must be manually hidden later by calling HidePopup( )

ShowInputPopup`( string mainText, string placeholderPrompt, string defaultInputText, string buttonText, UnityAction< string > callbackMethod )`

- Shows the popup prompting the user to input some text
- Placeholder prompt will display in the input field when empty
- Default input text is what will initially appear in the input field
- UnityAction callback will be called when button is pressed
- string callback parameter will contain the text that was input by the user

HidePopup`( )`

- Hides any active popup
- Generally only necessary for loading screen popups, others hide themselves when a button is pressed

ShowContextPopup`( RectTransform rt, Sprite[] buttonSprites, UnityAction< int > callbackMethod )`

- Shows a context popup that will be positioned at the pivot of the provided RectTransform
- Creates one graphical button for each Sprite provided
- int callback parameter will contain the 0-based index of the button that was pressed

HideContextPopup`( )`

- Hides an active context popup
- Context popups hide themselves when a button is pressed or when a click is registered outside the panel, but this may still be useful in some situations

Support and Feedback

Please don't hesitate to contact us at support@jarcas.com if you need any help with JPUI. Feedback is also encouraged. Anything you hate, love, or would like to see added/changed? Please let us know!

Follow Jarcas Studios on Twitter @JarcasStudios

Jarcas Studios
www.jarcas.com