Dialogue System for Unity Menu Framework

Version: 2017-01-09

This package contains a menu framework that leverages the Dialogue System for Unity.

Requirements

- Unity 5.3.0+
- Dialogue System for Unity 1.6.6.4+
- *Note: Previous versions of this menu framework required Unity's Game Jam Menu Template. This version no longer requires or uses the Game Jam Menu Template.*

Features

- Title Screen Main Menu:
    - Start/Continue/Load game
    - Options menu, Credits scene
- Gameplay Pause Menu:
    - Quests (open quest log window)
    - Save & Load games
    - Options menu
    - Quit to title screen or quit program
- Options Menu:
    - Set fullscreen/windowed, resolution, and graphics quality settings
    - Set music and sound effects volumes
    - Toggle subtitles
- General features:
    - Gracefully switches between joystick and keyboard/mouse modes, auto-focusing buttons and hiding the mouse cursor in joystick mode to allow proper navigation

Example Scene

To see the menu framework in action:

1. Open the Build Settings window.
2. Add these scenes:
   - Scene 0: Examples/Start
   - Scene 1: Examples/Loading
   - Scene 2: Examples/Credits
   - Scene 3: Gameplay
3. Play the Start scene.

Setup

To use this menu framework in your project:

1. Import the Dialogue System for Unity.

2. Import this package.

3. Add a **Dialogue Manager** to your title menu scene. Assign your dialogue database and dialogue UI to it. This will be your main Dialogue Manager. It will persist when you change levels.

4. Add a *LevelManager* component to your Dialogue Manager.

5. Add the prefab Dialogue System Menu Framework/Prefabs/**Menu System** as a child of the Dialogue Manager.

6. If you're using a splash scene before your title menu scene (e.g., Examples/Start), add the splash scene in build settings as scene 0 and the title menu scene as scene 1. Otherwise leave the title menu scene as scene 0.

7. If you're using a credits scene, add the credits scene as scene 3. Otherwise leave it as scene 2.

8. Inspect the **Menu System**. Set the correct scene indices in the *SaveHelper* and *TitleMenu* components.

9. Inspect the *InputDeviceManager* component. This component handles switching between joystick, keyboard, mouse, and touch input modes.
   - Click the **Add Input Definitions** button at the bottom of the component's inspector to add any missing input definitions to Unity's Input Manager.
   - Optionally customize the list of buttons and axes to check when determining which input device to use.

    - Note: If you use a different input system such as Rewired or InControl, you can assign a delegate method to InputDeviceManager.GetButtonDown.

10. Customize the placeholder images and text.
    - Activate each child panel of the Menu System so you can see it while editing. After editing, remember to deactivate it.
    - If the panel has an Animator with trigger parameters named “Show” and “Hide”, it will set those triggers when showing or hiding the panel. By default, the PausePanel has an *AnimatorController* that expands the window on Show and shrinks it on Hide.

11. Every panel has On Open and On Close events to which you can assign handlers. By default, when the PausePanel is open it sets Time.timeScale to zero and hides the player's *Selector/ProximitySelector* component (if it's in the scene).

12. The Menu System starts with three saved game slots. If you want to adjust the number of slots:
    - Activate the LoadGamePanel.
    - Duplicate or delete slot buttons.
    - Inspect each slot button and assign a unique slot number (e.g., 0, 1, 2, etc.) to On Click.
    - Repeat the process for the SaveGamePanel.

13. When loading games, the LoadInProgressPanel is shown. If you don't want to show this, unassign it from the Menu System's SaveHelper component. If you want to show a loading screen instead, inspect *SaveHelper* and tick Use Loading Scene. Then specify the build indices of the loading scene and the first gameplay scene. If you want to do something more, assign an event handler to LoadGamePanel's On Load Game event.

14. The *SaveHelper* component saves games to PlayerPrefs. If you want to save games to disk (desktop builds only), enable the *SaveToDisk* script on the Menu System. If you want to save another way, assign your own delegate methods to SaveHelper.SaveSlotHandler and SaveHelper.LoadSlotHandler. To enable quick-save and/or quick-load, set the button names on the SaveHelper component.

15. During gameplay, you can set a Dialogue System text variable named "CurrentStage". The contents of this variable will added to the saved game summary information, which is shown in the details section of the load game panel when the player selects a saved game. For example, you can use a *LuaTrigger* set to *OnStart* that sets Variable[“CurrentStage”] to the name of the scene, or use a dialogue entry’s Script field to set “CurrentStage” after major story events.

    If you want to add more information to the saved game summary, assign your own delegate methods to SaveHelper.RecordExtraSlotDetailsHandler.

    To show saved game details differently in the load game panel, you can assign an event handler to LoadGamePanel's On Set Details event.

16. To facilitate playtesting, you can add a Dialogue Manager with a Menu System child to your gameplay scenes. This way you can playtest them without having to come in from the title

menu scene. Just keep in mind that, when you come in from the title menu scene, the title menu scene's Dialogue Manager will be the active Dialogue Manager. In your gameplay scene, deactivate the Menu System's TitleMenuPanel.

Loading Scenes

If you've enabled loading scenes in the SaveHelper component, you can load a new scene using the new LoadingSceneTo() sequencer command:

LoadingSceneTo(*levelName*, [*loadingSceneIndex*], [*spawnpoint*])

Parameters:

- *levelName*: The name of the level to load asynchronously from the loading scene.
- *loadingSceneIndex*: The scene index of the loading scene to use. If omitted, uses the loading scene index specified in SaveHelper.
- *spawnpoint*: If specified, tells the player's Persistent Position Data to move the player to the position of the GameObject named *spawnpoint*. Assumes the player's actor in the dialogue database is named “Player”.

To load a new scene from a script, call SaveHelper.LoadLevel(levelName) or SaveHelper.LoadLevel(levelName, loadingSceneIndex).