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

/// <summary>
/// MenuInterfaceManager handles all UI interactions and transitions within the menu scene.
/// </summary>
public class MenuInterfaceManager : MonoBehaviour
{
    // Singleton state reference
    private static MenuInterfaceManager instance;

    /// <summary>
    /// The colors of save slots when uncreated and created. respectively.
    /// </summary>
    [SerializeField]
    Color defaultColor,
        saveColor;

    /// <summary>
    /// How much the height of <seealso cref="deleteErrorTransform"/> should increase for each additional error message.
    /// </summary>
    [SerializeField] float deleteErrorMessageSize;

    /// <summary>
    /// Attempts to exit the <seealso cref="currentInterface"/> in use.<br/><br/>
    /// For some interfaces, this will not instantly occur (such as going back to the options menu from the guide).
    /// </summary>
    [SerializeField] KeyCode cancelKey;

    /// <summary>
    /// Utilized to add and reference custom circuits from the directory.
    /// </summary>
    [SerializeField]
    GameObject customCircuitPanel,
        customCircuitPrefab;

    /// <summary>
    /// Set to visible when an interface is opened.<br/><br/>
    /// Within the scene, this should be a semi-transparent background overlayed onto everything except the currently opened interface.
    /// </summary>
    [SerializeField]
    GameObject transparentBackground,
```

```
45          transparentBackgroundUpper; // Utilized for multi-level interfaces
               such as the options interface.
46
47      [Space(10)]
48      /// <summary>
49      /// List of all interfaces within the scene.
50      /// </summary>
51      [SerializeField]
52      GameObject customCircuitsInterface, // Displays all currently open
          custom circuits and allows the user to view and/or delete them;
          opened from the options menu.
53          deleteErrorInterface, // The error log(s) displayed when a custom
              circuit cannot be deleted for any reason(s).
54          guideInterface, // Displays the guide prefab; opened from the
              options menu.
55          optionSelectionInterface, // Allows the user to activate
              customCircuitsInterface as well as guideInterface.
56          optionsInterface, // The parent of optionSelectionInterface
              indicating that the options menu is open.
57          sceneDeletionInterface, // Prompts the user to confirm deleting an
               editor structure (save slot).
58          sceneNameInterface; // Prompts the user to compose a name for an
              editor structure (save slot).
59
60      [Space(10)]
61      /// <summary>
62      /// Utilized to expand the vertical height of the error interface by
          factors of <seealso cref="deleteErrorMessageSize"/>.
63      /// </summary>
64      [SerializeField]
65      RectTransform deleteErrorTransform;
66
67      /// <summary>
68      /// The text components for the delete and scene creation interfaces
          utilized to display the cause(s) of error.
69      /// </summary>
70      [SerializeField]
71      TextMeshProUGUI deleteErrorText,
72      sceneNameError;
73
74      /// <summary>
75      /// The input field within <seealso cref="sceneNameInterface"/> that
          the user uses to compose a scene name.
76      /// </summary>
77      [SerializeField]
78      TMP_InputField sceneNameInputField;
79
80      /// <summary>
81      /// Text components of all 3 save slots within the menu used for
```

```
          setting and deleting their names.
 82       /// </summary>
 83       [SerializeField]
 84       TextMeshProUGUI save1,
 85           save2,
 86           save3;
 87
 88       /// <summary>
 89       /// The current UI interface in use.
 90       /// </summary>
 91       private GameObject currentInterface;
 92
 93       /// <summary>
 94       /// Index of the current <see cref="EditorStructure"/> loaded into the
             scene.
 95       /// </summary>
 96       private int currentSceneIndex = -1;
 97
 98       // Enforces a singleton state pattern
 99       private void Awake()
100       {
101           if (instance != null)
102           {
103               Destroy(this);
104               throw new Exception("MenuInterfaceManager instance already
                      established; terminating.");
105           }
106
107           instance = this;
108       }
109
110       // Loads in all serialized editor scenes and preview structures.
111       private void Start()
112       {
113           UpdateInterface();
114           AddCustomBookmarks();
115           CursorManager.SetMouseTexture(true);
116           enabled = false;
117       }
118
119       private void Update()
120       {
121           // Default exit controls for all interfaces except the options
                  interface.
122           if (currentInterface != optionsInterface)
123           {
124               if (Input.GetKeyDown(cancelKey) || Input.GetMouseButtonDown
                      (1)) CancelCurrentSubmission();
125           }
```

```
126
127         // Otherwise, the options interface is opened and should utilize a different transition scheme.
128         else if (Input.GetKeyDown(cancelKey) || Input.GetMouseButtonDown(1))
129         {
130             // If at the root, exit the interface.
131             if (optionSelectionInterface.activeSelf)
                CancelCurrentSubmission();
132
133             // If within the custom circuits interface whilst the delete error interface is not open, return to the root.
134             else if (customCircuitsInterface.activeSelf && !deleteErrorInterface.activeSelf)
135             {
136                 customCircuitsInterface.SetActive(false);
137                 optionSelectionInterface.SetActive(true);
138             }
139
140             // If within the guide interface, return to the root.
141             else if (guideInterface.activeSelf)
142             {
143                 guideInterface.SetActive(false);
144                 optionSelectionInterface.SetActive(true);
145             }
146
147             // If within the guide delete error interface, re-adjust background layers to make the custom circuit interface accessible.
148             else if (deleteErrorInterface.activeSelf)
149             {
150                 transparentBackgroundUpper.SetActive(false);
151                 transparentBackground.SetActive(true);
152                 deleteErrorInterface.SetActive(false);
153             }
154         }
155     }
156
157     /// <summary>
158     /// Opens an editor scene; called by pressing one of the valid save buttons.
159     /// </summary>
160     /// <param name="sceneIndex"></param>
161     public void OpenScene(int sceneIndex) { MenuLogicManager.Instance.OpenScene(sceneIndex); }
162
163     /// <summary>
164     /// Instantiates a <seealso cref="customCircuitPrefab"/> button for each preview structure.
```

```
165     /// </summary>
166     private void AddCustomBookmarks()
167     {
168         List<PreviewStructure> previewStructures =
              MenuSetupManager.Instance.PreviewStructures;
169
170         foreach (PreviewStructure previewStructure in previewStructures)
171         {
172             GameObject current = Instantiate(customCircuitPrefab,
                  customCircuitPanel.transform);
173
174             current.GetComponentInChildren<TextMeshProUGUI>().text =
                  previewStructure.Name;
175
176             // Adds the relevant listeners to ensure the custom circuit
                  can be deleted and viewed by their respective buttons.
177             current.GetComponent<CustomCircuitButtons>
                  ().DeleteButton.onClick.AddListener(delegate { DeletePreview
                  (current, previewStructure); });
178             current.GetComponent<CustomCircuitButtons>
                  ().ViewButton.onClick.AddListener(delegate { PreviewScene
                  (previewStructure); });
179         }
180     }
181
182     /// <summary>
183     /// Deletes a preview structure from the game and directory given it
          is not in use.
184     /// </summary>
185     /// <param name="button">The button assigned this method by a
          delegate.</param>
186     /// <param name="previewStructure">The preview structure to attempt
          deletion on.</param>
187     public void DeletePreview(GameObject button, PreviewStructure
          previewStructure)
188     {
189         // Obtains a list of all error messages in an attempt to delete
              the preview structure
190         List<string> errorMessages =
              MenuLogicManager.CanDeleteCustomCircuit(previewStructure);
191
192         // If there are no errors, proceed with deletion.
193         if (errorMessages.Count == 0)
194         {
195             MenuSetupManager.Instance.DeletePreviewStructure
                  (previewStructure);
196             Destroy(button);
197         }
198
```

```
199         // Otherwise, open the delete error interface and display errors.
200         else
201         {
202             transparentBackgroundUpper.SetActive(true);
203             transparentBackground.SetActive(false);
204             deleteErrorInterface.SetActive(true);
205             deleteErrorTransform.sizeDelta = new Vector2
                  (deleteErrorTransform.sizeDelta.x, deleteErrorMessageSize *
                  errorMessages.Count);
206             deleteErrorText.text = "";
207 
208             int index = 0;
209 
210             // Adds each error
211             foreach (string errorMessage in errorMessages)
212             {
213                 index++;
214                 deleteErrorText.text += "- " + errorMessage + (index !=
                      errorMessages.Count ? "\n\n" : "");
215             }
216         }
217     }
218 
219     /// <summary>
220     /// Referenced when the user acknowledges and closes the delete error
          interface; called by pressing an in-scene button.
221     /// </summary>
222     public void OnCircuitDeleteErrorConfirm()
223     {
224         transparentBackgroundUpper.SetActive(false);
225         transparentBackground.SetActive(true);
226         deleteErrorInterface.SetActive(false);
227     }
228 
229     /// <summary>
230     /// Opens a preview structure whose internal components the user
          wishes to inspect.<br/><br/>
231     /// This method is assigned as a listener to the delete button within
          each instantiated <seealso cref="customCircuitPrefab"/>.
232     /// </summary>
233     /// <param name="previewStructure">The preview structure to open.</
          param>
234     public void PreviewScene(PreviewStructure previewStructure)
          { MenuLogicManager.Instance.OpenPreview(previewStructure); }
235 
236     /// <summary>
237     /// Sets the names of all save slots corresponding to serialized
          editor structures.
238     /// </summary>
```

```
239     public void UpdateInterface()
240     {
241         EditorStructure[] editorStructures =
              MenuSetupManager.Instance.EditorStructures;
242
243         if (editorStructures[0] != null) { save1.text = editorStructures
              [0].Name; save1.color = saveColor; }
244
245         if (editorStructures[1] != null) { save2.text = editorStructures
              [1].Name; save2.color = saveColor; }
246
247         if (editorStructures[2] != null) { save3.text = editorStructures
              [2].Name; save3.color = saveColor; }
248     }
249
250     /// <summary>
251     /// Opens <seealso cref="sceneDeletionInterface"/> prompting a user to
          acknowledge their action to delete a editor structure (save slot).
252     /// </summary>
253     /// <param name="sceneIndex">The index of the prospective editor
          structure to delete</param>
254     public void BeginSceneDeletion(int sceneIndex)
255     {
256         if (MenuSetupManager.Instance.EditorStructures[sceneIndex] ==
              null) return;
257
258         currentSceneIndex = sceneIndex;
259         BeginInterface(sceneDeletionInterface);
260     }
261
262     /// <summary>
263     /// Opens <seealso cref="sceneNameInterface"/> prompting a user to
          compose a name to create an editor structure.
264     /// </summary>
265     /// <param name="sceneIndex">The index of the prospective editor
          structure to create.</param>
266     public void BeginSceneNameSubmission(int sceneIndex)
267     {
268         currentSceneIndex = sceneIndex;
269         BeginInterface(sceneNameInterface);
270     }
271
272     /// <summary>
273     /// Opens the options interface; called by pressing an in-scene
          button.
274     /// </summary>
275     public void OpenOptionsInterface()
276     {
277         BeginInterface(optionsInterface);
```

```
278         optionSelectionInterface.SetActive(true);
              customCircuitsInterface.SetActive(false);
              guideInterface.SetActive(false);
279     }
280
281     /// <summary>
282     /// Opens the guide interface; called by pressing an in-scene button.
283     /// </summary>
284     public void OpenGuide()
285     {
286         optionSelectionInterface.SetActive(false);
287         guideInterface.SetActive(true);
288     }
289
290     /// <summary>
291     /// Opens the custom circuit interface; called by pressing an in-scene
          button.
292     /// </summary>
293     public void OpenCustomCircuits()
294     {
295         optionSelectionInterface.SetActive(false);
296         customCircuitsInterface.SetActive(true);
297     }
298
299     /// <summary>
300     /// Restores the text of a save slot to its default values after a
        success editor scene deletion.
301     /// </summary>
302     public void SceneDeleteSubmission()
303     {
304         MenuSetupManager.Instance.DeleteEditorStructure
              (currentSceneIndex);
305         TextMeshProUGUI currentText;
306
307         if (currentSceneIndex == 0)
308         {
309             currentText = save1;
310         }
311
312         else if (currentSceneIndex == 1)
313         {
314             currentText = save2;
315         }
316
317         else
318         {
319             currentText = save3;
320         }
321
```

```
322         CancelCurrentSubmission();
323         currentText.text = "new save";
324         currentText.color = defaultColor;
325 
326     }
327 
328     /// <summary>
329     /// Checks the name in <seealso cref="sceneNameInputField"/> and creates an editor scene if it is valid.
330     /// </summary>
331     public void SceneNameSubmission()
332     {
333         string submission = sceneNameInputField.text.ToLower().Trim();
334 
335         // Cannot be empty
336         if (submission == string.Empty) sceneNameError.text = "scene name must be non-empty";
337 
338         // Must be unique
339         else if (CurrentSceneNames.Contains(submission)) sceneNameError.text = "scene name must be unique";
340 
341         // Creates and opens the editor structure.
342         else MenuLogicManager.Instance.CreateScene(currentSceneIndex, submission);
343     }
344 
345     /// <summary>
346     /// Closes the current interface in use.
347     /// </summary>
348     public void CancelCurrentSubmission()
349     {
350         if (currentInterface == sceneNameInterface) sceneNameInputField.text = sceneNameError.text = "";
351 
352         BackgroundParallax.Instance.enabled = true;
353         ToggleCurrentInterface(false);
354         currentSceneIndex = -1;
355         currentInterface = null;
356         enabled = false;
357     }
358 
359     // Exits the game; called by pressing an in-scene button.
360     public void Quit() { Application.Quit(); }
361 
362     /// <summary>
363     /// Opens an interface.
364     /// </summary>
365     /// <param name="newInterface">The interface to open.</param>
```

```
366     private void BeginInterface(GameObject newInterface)
367     {
368         if (currentInterface != null) return;
369 
370         BackgroundParallax.Instance.enabled = false;
371         currentInterface = newInterface;
372         ToggleCurrentInterface(true);
373         enabled = true;
374     }
375 
376     /// <summary>
377     /// Sets the visibility of the current interface.
378     /// </summary>
379     /// <param name="visible">Whether the current interface should be
        visible.</param>
380     private void ToggleCurrentInterface(bool visible)
381     {
382         currentInterface.SetActive(visible);
          transparentBackground.SetActive(visible);
383     }
384 
385     // Getter methods
386     public static MenuInterfaceManager Instance { get { return
        instance; } }
387 
388     /// <summary>
389     /// Returns the named list of all editor structures.
390     /// </summary>
391     private List<string> CurrentSceneNames
392     {
393         get
394         {
395             EditorStructure[] editorStructures =
                MenuSetupManager.Instance.EditorStructures;
396             List<string> currentSceneNames = new List<string>();
397 
398             foreach (EditorStructure editorStructure in editorStructures)
                if (editorStructure != null) currentSceneNames.Add
                (editorStructure.Name);
399 
400             return currentSceneNames;
401         }
402     }
403 }
```