

```
1 using System;
2 using System.Collections.Generic;
3 using TMPro;
4 using UnityEditor;
5 using UnityEngine;
6 using UnityEngine.EventSystems;
7 using UnityEngine.SceneManagement;
8 using UnityEngine.UI;
9
10 public class TaskbarManager : MonoBehaviour
11 {
12     // Singleton state reference
13     private static TaskbarManager instance;
14
15     /// <summary>
16     /// The color associated with starting and custom circuits
17     /// </summary>
18     [SerializeField]
19     Color startingCircuitColor,
20         customCircuitColor;
21
22     /// <summary>
23     /// Horizontal length of the scroll bar attached to the bookmarks
24     /// </summary>
25     [SerializeField]
26     float bookmarkScrollThickness;
27
28     /// <summary>
29     /// Set to visible when an interface is opened.<br/><br/>
30     /// Within the scene, this should be a semi-transparent background
31     /// </summary>
32     [SerializeField]
33     GameObject background;
34
35     /// <summary>
36     /// The scroll bar belonging to the bookmarks menu.<br/><br/>
37     /// If the size of bookmarks exceeds the view area of the bookmarks
38     /// </summary>
39     [SerializeField]
40     GameObject bookmarkScrollbar;
41
42     /// <summary>
43     /// GameObject that all bookmarks are instantiated under.
44     /// </summary>
45     [SerializeField]
```

```

46     GameObject bookmarksPanel;
47
48     /// <summary>
49     /// Referenced when moving the bookmarks menu to the current user mouse position.
50     /// </summary>
51     [SerializeField]
52     GameObject bookmarksScroll;
53
54     /// <summary>
55     /// List of all menus that can be active in the editor scene.
56     /// </summary>
57     [SerializeField]
58     GameObject addMenu, // The menu in which the user can add/bookmark circuits.
59     bookmarksMenu, // The menu that displays bookmarked circuits, if any.
60     circuitSaveErrorMenu, // The menu indicating the reason a custom circuit has failed to create.
61     guide, // The guide prefab.
62     labelMenu, // The menu where the user labels empty inputs/outputs.
63     notifierPanel, // An empty menu without a user-driven exit scheme; has UI indicating why (e.g. saving).
64     nullState, // An empty menu without a user-driven exit scheme; does not have UI.
65     saveWarning, // The menu prompting the user to save.
66     sceneSaveMenu; // The menu in which the user can either save the editor scene or create a custom circuit.
67
68     /// <summary>
69     /// Prefab button for custom circuits within the add menu.
70     /// </summary>
71     [SerializeField]
72     GameObject customBookmarkRef;
73
74     /// <summary>
75     /// Prefab button for any circuit within the bookmarks menu.
76     /// </summary>
77     [SerializeField]
78     GameObject bookmarkRef;
79
80     /// <summary>
81     /// Exits out of <seealso cref="currentMenu"/>.<br/><br/>
82     /// More often than not, an alternate input to achieve the same effect is pressing the right mouse button.
83     /// </summary>
84     [SerializeField]
85     KeyCode cancelKey;
86

```

```
87     /// <summary>
88     /// Parent of all starting and custom circuits buttons within the add menu.
89     /// </summary>
90     [SerializeField]
91     RectTransform addCustomPanel,
92         addStartingPanel;
93
94     /// <summary>
95     /// Transform of the border behind the bookmarks menu.
96     /// </summary>
97     [SerializeField]
98     RectTransform bookmarksBorder;
99
100    /// <summary>
101    /// Displays the reason for a custom circuit failing to create.
102    /// </summary>
103    [SerializeField]
104    TextMeshProUGUI circuitErrorText;
105
106    /// <summary>
107    /// Prompts the user to compose a label for an empty input or output.
108    /// </summary>
109    [SerializeField]
110    TextMeshProUGUI labelText;
111
112    /// <summary>
113    /// Utilized with <seealso cref="notifierPanel"/> to display the reason why a game has disabled all input to the player.<br/><br/>
114    /// Its primary uses are for when the game is saving as well as when a custom circuit is being verified/created.
115    /// </summary>
116    [SerializeField]
117    TextMeshProUGUI notifierText;
118
119    /// <summary>
120    /// The name field with which the user specifies the name of a prospective custom circuit.
121    /// </summary>
122    [SerializeField]
123    TMP_InputField circuitNameField;
124
125    /// <summary>
126    /// Whether a custom circuit should be created.<br/>
127    /// If unchecked, then the current editor scene is saved instead.
128    /// </summary>
129    [SerializeField]
130    Toggle circuitToggle;
131
```

```
132     /// <summary>
133     /// The size of the bookmark view area and a bookmark respectively.
134     /// </summary>
135     [SerializeField]
136     Vector2 bookmarkMaskSize,
137         bookmarkSize;
138
139     /// <summary>
140     /// Whether the left mouse button is currently held down whilst in the
141     /// bookmarks menu.<br/>
142     /// This helps discern whether the initial left mouse button press and
143     /// release occurred when hovered on UI elements.
144     /// </summary>
145     private bool bookmarksDown;
146
147     /// <summary>
148     /// Whether the game is currently deserializing all bookmarks
149     /// belonging to the current editor scene.
150     /// </summary>
151     private bool currentlyRestoring;
152
153     /// <summary>
154     /// Whether the bookmarks bar can be opened.<br/>
155     /// This value is false until the cooldown to open the bookmarks bar
156     /// passes, enabling it again.
157     /// </summary>
158     private bool reopenBookmarks = true;
159
160     /// <summary>
161     /// The menu currently opened within the editor scene.
162     /// </summary>
163     private GameObject currentMenu;
164
165     /// <summary>
166     /// The ID list of bookmarks in the scene.<br/><br/>
167     /// Helps to differentiate whether the bookmark is a starting or
168     /// custom circuit, since all starting circuits have an ID of -1.
169     /// </summary>
170     private List<int> bookmarkIDs = new List<int>();
171
172     /// <summary>
173     /// The typed list of bookmarks in the scene.
174     /// </summary>
175     private List<Type> bookmarks = new List<Type>();
176
177     // Enforces a singleton state pattern and disables frame-by-frame
178     // update calls.
179     private void Awake()
180     {
```

```
175         if (instance != null)
176         {
177             Destroy(this);
178             throw new Exception("TaskbarManager instance already established; terminating.");
179         }
180
181         instance = this;
182         enabled = false;
183     }
184
185     // Contains input listening for each user-controllable control interface.
186     private void Update()
187     {
188         // No menu is currently opened, skips current frame
189         if (currentMenu == nullState) return;
190
191         // Bookmark control scheme
192         if (currentMenu == bookmarksMenu)
193         {
194             // Registers left mouse button down
195             if (Input.GetMouseButtonDown(0) && !
196                 EventSystem.current.IsPointerOverGameObject())
197             {
198                 bookmarksDown = true;
199             }
200
201             // Exit scheme (occurs if left mouse button is released, but
202             // not while hovered on UI.
203             else if (Input.GetMouseButtonUp(0) && bookmarksDown)
204             {
205                 if (EventSystem.current.IsPointerOverGameObject())
206                     bookmarksDown = false; else CloseMenu();
207             }
208
209             // Moves bookmark to new mouse position
210             else if (Input.GetMouseButtonDown(1) && !
211                 EventSystem.current.IsPointerOverGameObject())
212             {
213                 UpdateBookmarkPosition();
214             }
215
216             // Keyboard exit scheme
217             else if (Input.GetKeyDown(cancelKey)) CloseMenu();
218         }
219
220         // Default control scheme for all other menus allowed to be
221         // existed by the user.
222         else if (currentMenu == addMenu || currentMenu == sceneSaveMenu ||
223             currentMenu == circuitSaveErrorMenu || currentMenu ==
224             saveWarning || currentMenu == guide)
225         {
226         }
```

```

...object\Assets\Scripts\Shared Scripts\TaskbarManager.cs 6
213         if (Input.GetKeyDown(cancelKey) || Input.GetMouseButtonDown  ↗
            (1))
214         {
215             if (currentMenu == circuitSaveErrorMenu) ConfirmError();  ↗
                else CloseMenu();
216         }
217     }
218 }
219
220 /// <summary>
221 /// Opens the <seealso cref="nullState"/> interface.<br/>
222 /// Essentially disables the taskbar from functioning; a locked state  ↗
    that must be manually closed via script.
223 /// </summary>
224 public void NullState() { OpenMenu(false, nullState); }
225
226 /// <summary>
227 /// Updates <seealso cref="circuitSaveErrorMenu"/> after <seealso  ↗
    cref="circuitToggle"/> is pressed; called by pressing an in-scene  ↗
    button.
228 /// </summary>
229 public void UpdateSaveToggle()
230 {
231     bool isOn = circuitToggle.isOn;
232
233     circuitNameField.interactable = isOn;
234
235     if (!isOn) circuitNameField.text = "";
236 }
237
238 /// <summary>
239 /// Goes back to the menu; called by pressing an in-scene button.
240 /// </summary>
241 public void OpenOptions()
242 {
243     // If the current scene is in the editor, check if the save prompt  ↗
        should first be displayed. Otherwise (including if in a preview  ↗
        scene), go back to the menu.
244     if (EditorStructureManager.Instance != null &&  ↗
        EditorStructureManager.Instance.DisplaySavePrompt) OpenMenu  ↗
        (true, saveWarning); else SceneManager.LoadScene(0);
245 }
246
247 /// <summary>
248 /// Goes back to the game menu; called by pressing an in-scene button.
249 /// </summary>
250 public void OpenMenuScene() { SceneManager.LoadScene(0); }
251
252 /// <summary>

```

```

...object\Assets\Scripts\Shared Scripts\TaskbarManager.cs 7
253  /// Opens <seealso cref="labelMenu"/> <see cref="IOAssigner"/> is  ↗
    enabled and the user presses LMB on an incomplete empty input/  ↗
    output.
254  /// </summary>
255  /// <param name="isInput"></param>
256  public void OpenLabelMenu(bool isInput)
257  {
258      OpenMenu(true, labelMenu);
259      labelText.text = "compose a label for the selected " + (isInput ?  ↗
        "input" : "output");
260  }
261
262  /// <summary>
263  /// Opens <seealso cref="guide"/>; called by pressing an in-scene  ↗
    button.
264  /// </summary>
265  public void OpenGuide() { OpenMenu(true, guide); }
266
267  /// <summary>
268  /// Opens <seealso cref="sceneSaveMenu"/>; called by pressing an in-  ↗
    scene button.
269  /// </summary>
270  public void OpenSave() { OpenMenu(true, sceneSaveMenu); }
271
272  /// <summary>
273  /// Displays an error message if saving a custom circuit fails.
274  /// </summary>
275  /// <param name="errorMessage">The error message to display.</param>
276  public void CircuitSaveError(string errorMessage)
277  {
278      CloseMenu();
279      circuitErrorText.text = errorMessage;
280      OpenMenu(true, circuitSaveErrorMenu);
281  }
282
283  /// <summary>
284  /// Closes <seealso cref="circuitSaveErrorMenu"/>; can be called by  ↗
    pressing an in-scene button.
285  /// </summary>
286  public void ConfirmError()
287  {
288      CloseMenu();
289      OpenSave();
290  }
291
292  /// <summary>
293  /// Confirms <seealso cref="sceneSaveMenu"/> input and either saves  ↗
    the editor scene or creates a custom circuit based on <seealso  ↗
    cref="circuitToggle"/>.<br/>

```

```
294     /// Called by pressing an in-scene button.
295     /// </summary>
296     public void SaveConfirm()
297     {
298         CloseMenu();
299
300         // Should attempt to create custom circuit
301         if (circuitToggle.isOn)
302         {
303             notifierText.text = "verifying...";
304             OpenMenu(true, notifierPanel);
305             PreviewStructureManager.Instance.VerifyPreviewStructure      ↗
                 (circuitNameField.text.ToLower().Trim());
306         }
307
308         // Should save scene
309         else
310         {
311             notifierText.text = "saving scene...";
312             OpenMenu(true, notifierPanel);
313             EditorStructureManager.Instance.Serialize();
314         }
315     }
316
317     /// <summary>
318     /// Opens <seealso cref="addMenu"/>; called by pressing an in-scene      ↗
319     /// button.
320     /// </summary>
321     public void OpenAdd() { OpenMenu(true, addMenu); }
322
323     /// <summary>
324     /// Opens the bookmarks menu.
325     /// </summary>
326     public void OpenBookmarks() { OpenBookmarks(false); }
327
328     /// <summary>
329     /// Called when a custom circuit is successfully created.
330     /// </summary>
331     public void OnSuccessfulPreviewStructure()
332     {
333         circuitNameField.text = "";
334         CloseMenu();
335     }
336
337     /// <summary>
338     /// Called when a custom circuit successfully passes validation.
339     /// </summary>
340     public void OnSuccessfulPreviewVerification()
341     {
```



```
341         CloseMenu();
342         OpenMenu(true, notifierPanel);
343         notifierText.text = "creating...";
344     }
345
346     /// <summary>
347     /// Opens the bookmarks menu.<br/>
348     /// If there are no bookmarks to display, this method does nothing.
349     /// </summary>
350     /// <param name="showBackground"></param>
351     public void OpenBookmarks(bool showBackground)
352     {
353         if (bookmarks.Count == 0) return;
354
355         bookmarksDown = false;
356         OpenMenu(showBackground, bookmarksMenu);
357     }
358
359     /// <summary>
360     /// Deserializes all bookmarks stored in the current editor structure.
361     /// </summary>
362     /// <param name="circuitIndeces">The serialized integer to circuit
363     identifiers.</param>
364     /// <param name="circuitIDs">The preview structure IDs of each circuit
365     (-1 if non-custom).</param>
366     public void RestoreBookmarks(List<int> circuitIndeces, List<int>
367     circuitIDs)
368     {
369         int index = 0;
370
371         currentlyRestoring = true;
372
373         foreach (int circuitIndex in new List<int>(circuitIndeces))
374         {
375             // Is a custom circuit
376             if (circuitIndex != -1)
377             {
378                 Toggle toggle = addStartingPanel.GetChild
379                 (circuitIndex).GetComponentInChildren<Toggle>();
380
381                 toggle.isOn = true;
382                 UpdateBookmarkAll(toggle.gameObject);
383             }
384
385             // Is a starting circuit
386             else AddCustomCircuitPanel(circuitIDs[index], true);
387
388             index++;
389         }
390     }
```

```
386
387     currentlyRestoring = false;
388 }
389
390 /// <summary>
391 /// Adds all non-bookmarked custom circuits belonging to the current preview structure back to <seealso cref="addMenu"/>.
392 /// </summary>
393 public void RestoreCustomCircuits()
394 {
395     foreach (PreviewStructure previewStructure in MenuSetupManager.Instance.PreviewStructures)
396     {
397         // Is bookmarked, continue.
398         if (bookmarkIDs.Contains(previewStructure.ID)) continue;
399
400         AddCustomCircuitPanel(previewStructure.ID, false);
401     }
402 }
403
404 /// <summary>
405 /// Adds a custom circuit to <seealso cref="addCustomPanel"/>.
406 /// </summary>
407 /// <param name="circuitID">The custom circuit ID.</param>
408 /// <param name="bookmarked">Whether this circuit is bookmarked in the current editor scene.</param>
409 public void AddCustomCircuitPanel(int circuitID, bool bookmarked)
410 {
411     GameObject current = Instantiate(customBookmarkRef, addCustomPanel.transform); // Instantiates a prefab copy
412     Toggle toggle = current.GetComponentInChildren<Toggle>();
413     PreviewStructure.PreviewStructureReference reference = current.AddComponent<PreviewStructure.PreviewStructureReference>();
414
415     current.GetComponentInChildren<TextMeshProUGUI>().text = MenuSetupManager.Instance.PreviewStructures[MenuSetupManager.Instance.PreviewStructureIDs.IndexOf(circuitID)].Name;
416     reference.ID = circuitID;
417
418     // Adds listeners required to add and bookmark the custom circuit.
419     current.GetComponentInChildren<Button>().onClick.AddListener(delegate { AddBookmarkCircuit(-1, reference.ID); });
420     toggle.onValueChanged.AddListener(delegate { UpdateBookmark(reference); });
421
422     if (bookmarked)
423     {
```

```
424         toggle.isOn = true;
425         UpdateBookmarkCustom(reference);
426     }
427 }
428
429 /// <summary>
430 /// Identical to <seealso cref="UpdateBookmarkAll(GameObject)"/> except reserved specifically for <see cref="Toggle"/> calls.<br/>
431 /// This is due to boolean changes within scripting also triggering <seealso cref="Toggle.onValueChanged"/> events.
432 /// </summary>
433 /// <param name="obj">The bookmark to update.</param>
434 public void UpdateBookmark(GameObject obj)
435 {
436     // If toggles are being adjusted within the script, do not
437     // continue.
438     if (currentlyRestoring) return;
439     UpdateBookmarkAll(obj);
440 }
441
442 /// <summary>
443 /// Identical to <seealso cref="UpdateBookmarkCustom
444     (PreviewStructure.PreviewStructureReference)"/> except reserved
445     specifically for <see cref="Toggle"/> calls.<br/><br/>
446     This is due to boolean changes within scripting also triggering
447     <seealso cref="Toggle.onValueChanged"/> events.
448     </summary>
449     <param name="obj">The bookmark to update.</param>
450 public void UpdateBookmark(PreviewStructure.PreviewStructureReference
451     previewStructureReference)
452 {
453     // If toggles are being adjusted within the script, do not
454     // continue.
455     if (currentlyRestoring) return;
456     UpdateBookmarkCustom(previewStructureReference);
457 }
458
459 /// <summary>
460 /// Updates a starting bookmark after it has been bookmarked or
461     unbookmarked.
462     </summary>
463     <param name="obj">The starting bookmark to update.</param>
464 public void UpdateBookmarkAll(GameObject obj)
465 {
466     bool newStatus = obj.GetComponent<Toggle>().isOn;
467     Type type = CircuitType(obj.transform.parent.GetSiblingIndex());
```

```
463
464     // Adds bookmark
465     if (newStatus && !bookmarks.Contains(type))
466     {
467         if (!currentlyRestoring) ↗
468             EditorStructureManager.Instance.DisplaySavePrompt = true;
469
470         EditorStructureManager.Instance.Bookmarks.Add ↗
471             (StartingCircuitIndex(type));
472         bookmarks.Add(type);
473         bookmarkIDs.Add(-1);
474
475         GameObject bookmark = Instantiate(bookmarkRef, ↗
476             bookmarksPanel.transform);
477         Button button = bookmark.GetComponentInChildren<Button>();
478         TextMeshProUGUI text = ↗
479             bookmark.GetComponentInChildren<TextMeshProUGUI>();
480
481         bookmark.name = text.text = obj.transform.parent.name;
482         text.color = startingCircuitColor;
483
484         // Ensures pressing on the bookmark will add its ↗
485         representative circuit.
486         button.onClick.AddListener(delegate { AddBookmarkCircuit ↗
487             (StartingCircuitIndex(type), -1); });
488     }
489
490     // Deletes bookmark
491     else if (!newStatus && bookmarks.Contains(type))
492     {
493         if (!currentlyRestoring) ↗
494             EditorStructureManager.Instance.DisplaySavePrompt = true;
495
496         int index = bookmarks.IndexOf(type);
497
498         EditorStructureManager.Instance.Bookmarks.Remove ↗
499             (StartingCircuitIndex(type));
500         bookmarks.Remove(type);
501         bookmarkIDs.RemoveAt(index);
502         Destroy(bookmarksPanel.transform.GetChild(index).gameObject);
503     }
504 }
505
506 /// <summary>
507 /// Updates a custom bookmark after it has been bookmarked or ↗
508 unbookmarked.
509 /// </summary>
510 /// <param name="obj">The custom bookmark to update.</param>
511 public void UpdateBookmarkCustom ↗
```

```
(PreviewStructure.PreviewStructureReference reference)
503     {
504         bool newStatus = reference.GetComponentInChildren<Toggle>().isOn;
505         int id =
            reference.GetComponentInChildren<PreviewStructure.PreviewStructu
            reReference>().ID;

506
507         // Adds bookmark
508         if (newStatus && !bookmarkIDs.Contains(id))
509         {
510             if (!currentlyRestoring)
511                 EditorStructureManager.Instance.DisplaySavePrompt = true;
512
513             EditorStructureManager.Instance.Bookmarks.Add(-1);
514             bookmarks.Add(typeof(CustomCircuit));
515             bookmarkIDs.Add(id);
516
517             GameObject bookmark = Instantiate(bookmarkRef,
518                 bookmarksPanel.transform);
519             Button button = bookmark.GetComponentInChildren<Button>();
520             TextMeshProUGUI text =
521                 bookmark.GetComponentInChildren<TextMeshProUGUI>();
522
523             bookmark.name = text.text =
524                 MenuSetupManager.Instance.PreviewStructures
525                 [MenuSetupManager.Instance.PreviewStructureIDs.IndexOf
526                 (id)].Name;
527             text.color = customCircuitColor;
528
529             // Ensures pressing on the bookmark will add its
530             // representative circuit.
531             button.onClick.AddListener(delegate { AddBookmarkCircuit(-1,
532                 id); });
533         }
534
535         // Deletes bookmark
536         else if (!newStatus && bookmarkIDs.Contains(id))
537         {
538             if (!currentlyRestoring)
539                 EditorStructureManager.Instance.DisplaySavePrompt = true;
540
541             int index = bookmarkIDs.IndexOf(id);
542
543             EditorStructureManager.Instance.Bookmarks.RemoveAt(index);
544             bookmarks.RemoveAt(index);
545             bookmarkIDs.Remove(id);
546             Destroy(bookmarksPanel.transform.GetChild(index).gameObject);
547         }
548     }
549 }
```

```
540
541     /// <summary>
542     /// Adds a bookmarked circuit based on its circuit type and custom      ↗
543     circuit ID (if applicable).
544     /// </summary>
545     /// <param name="circuitType">Index representing the circuit type.</    ↗
546     param>
547     /// <param name="circuitID">The custom circuit ID (-1 if custom).</    ↗
548     param>
549     private void AddBookmarkCircuit(int circuitType, int circuitID)
550     {
551         switch (circuitType)
552         {
553             // Custom circuit
554             case -1:
555                 AddCircuit(new CustomCircuit
556                     (MenuSetupManager.Instance.PreviewStructures
557                     [MenuSetupManager.Instance.PreviewStructureIDs.IndexOf
558                     (circuitID)]));
559                 return;
560             // Input
561             case 0:
562                 AddCircuit(new InputGate());
563                 return;
564             // Display
565             case 1:
566                 AddCircuit(new Display());
567                 return;
568             // Buffer
569             case 2:
570                 AddCircuit(new Buffer());
571                 return;
572             // And gate
573             case 3:
574                 AddCircuit(new AndGate());
575                 return;
576             // NAnd gate
577             case 4:
578                 AddCircuit(new NAndGate());
579                 return;
580             // NOr gate
581             case 5:
582                 AddCircuit(new NOrGate());
583                 return;
584             // Not gate
585             case 6:
586                 AddCircuit(new NotGate());
587                 return;
588             // Or gate
```

```
583         case 7:
584             AddCircuit(new OrGate());
585             return;
586         // XOr gate
587         case 8:
588             AddCircuit(new XOrGate());
589             return;
590     }
591 }
592
593 /// <summary>
594 /// Adds a starting circuit to the scene; called by pressing an in- ➤
595   scene button.
596 /// </summary>
597 /// <param name="startingCircuitIndex">Representative index of the ➤
598   starting circuit.</param>
599 public void AddStartingCircuit(int startingCircuitIndex) { AddCircuit ➤
600   (GetStartingCircuit(startingCircuitIndex)); }
601
602 /// <summary>
603 /// Adds a circuit to the scene.
604 /// </summary>
605 /// <param name="newCircuit">The circuit to add.</param>
606 private void AddCircuit(Circuit newCircuit)
607 {
608     // Cancels any modes that would obstruct the placement process.
609     switch (BehaviorManager.Instance.UnpausedGameState)
610     {
611         case BehaviorManager.GameState.CIRCUIT_MOVEMENT:
612             BehaviorManager.Instance.CancelCircuitMovement();
613             break;
614
615         case BehaviorManager.GameState.IO_PRESS:
616             BehaviorManager.Instance.CancelWirePlacement();
617             break;
618     }
619
620     // Switches to placement mode
621     BehaviorManager.Instance.UnpausedGameState = ➤
622       BehaviorManager.GameState.CIRCUIT_PLACEMENT;
623     BehaviorManager.Instance.UnpausedStateType = ➤
624       BehaviorManager.StateType.LOCKED;
625     BehaviorManager.Instance.CircuitPlacement(newCircuit);
626     CloseMenu();
627 }
628
629 /// <summary>
630 /// Obtains the circuit type based on its circuit index.
631 /// </summary>
```

```
627    /// <param name="circuitIndex">The index of the circuit.</param>
628    /// <returns>The type of the circuit.</returns>
629    private Type CircuitType(int circuitIndex)
630    {
631        switch (circuitIndex)
632        {
633            case -1:
634                return typeof(CustomCircuit);
635            case 0:
636                return typeof(InputGate);
637            case 1:
638                return typeof(Display);
639            case 2:
640                return typeof(Buffer);
641            case 3:
642                return typeof(AndGate);
643            case 4:
644                return typeof(NAndGate);
645            case 5:
646                return typeof(NOrGate);
647            case 6:
648                return typeof(NotGate);
649            case 7:
650                return typeof(OrGate);
651            case 8:
652                return typeof(XOrGate);
653            default:
654                throw new Exception("Invalid starting circuit index.");
655        }
656    }
657
658    /// <summary>
659    /// Obtains the index representation of a starting circuit.
660    /// </summary>
661    /// <param name="circuitType">The type of the starting circuit.</param>
662    /// <returns>The index representation of the circuit.</returns>
663    private int StartingCircuitIndex(Type circuitType)
664    {
665        if (circuitType == typeof(InputGate)) return 0;
666
667        else if (circuitType == typeof(Display)) return 1;
668
669        else if (circuitType == typeof(Buffer)) return 2;
670
671        else if (circuitType == typeof(AndGate)) return 3;
672
673        else if (circuitType == typeof(NAndGate)) return 4;
674
```



```
675         else if (circuitType == typeof(NOrGate)) return 5;
676
677         else if (circuitType == typeof(NotGate)) return 6;
678
679         else if (circuitType == typeof(OrGate)) return 7;
680
681         else if (circuitType == typeof(XOrGate)) return 8;
682
683         else throw new Exception("Invalid starting circuit type.");
684     }
685
686     /// <summary>
687     /// Opens a menu.
688     /// </summary>
689     /// <param name="showBackground">Whether <seealso cref="background"/> ↗
690     ///     should be visible.</param>
691     /// <param name="newMenu">The menu to open.</param>
692     private void OpenMenu(bool showBackground, GameObject newMenu)
693     {
694         // If another menu is open, do nothing.
695         if (currentMenu != null && currentMenu != bookmarksMenu) return;
696
697         // Close the bookmarks menu if another menu is opened.
698         if (currentMenu == bookmarksMenu) CloseMenu();
699
700         currentMenu = newMenu;
701
702         // If applicable, the bookmarks menu should open around the user's ↗
703         // cursor.
704         if (newMenu == bookmarksMenu)
705         {
706             UpdateBookmarkPosition();
707             UpdateBookmarkScroll();
708         }
709
710         BehaviorManager.Instance.LockUI = true;
711         background.SetActive(showBackground); currentMenu.SetActive(true);
712         enabled = true; // Enables the frame-by-frame listener.
713     }
714
715     /// <summary>
716     /// Updates the size of the bookmarks menu and enables/disables the ↗
717     /// scroll bar.
718     /// </summary>
719     private void UpdateBookmarkScroll()
720     {
721         // If the bookmarks menu does not show all bookmarked circuits, the ↗
722         // vertical scroll bar should appear.
723         bool exceededViewport = bookmarkSize.y * bookmarks.Count > ↗
```

```
bookmarkMaskSize.y;

720
721     if (exceededViewport)
722     {
723         // If large enough to scroll, always starts at top of options list
724         bookmarksPanel.GetComponent<RectTransform>().anchoredPosition
            *= Vector2.right;
725         bookmarksPanel.GetComponent<RectTransform>().sizeDelta =
            Vector2.right * (bookmarkSize.x + bookmarkScrollThickness);
726         bookmarksBorder.sizeDelta = new Vector2(bookmarkSize.x +
            bookmarkScrollThickness, Mathf.Clamp(bookmarks.Count *
            bookmarkSize.y, 0, bookmarkMaskSize.y));
727     }
728
729     // Do not show scroll bar, all bookmarks are visible in the view area.
730     else
731     {
732         bookmarksPanel.GetComponent<RectTransform>().sizeDelta =
            Vector2.right * bookmarkSize.x;
733         bookmarksBorder.sizeDelta = new Vector2(bookmarkSize.x,
            Mathf.Clamp(bookmarks.Count * bookmarkSize.y, 0,
            bookmarkMaskSize.y));
734     }
735
736     bookmarkScrollbar.SetActive(exceededViewport);
737 }
738
739 /// <summary>
740 /// Moves the bookmarks menu to the current position of the mouse.
741 /// </summary>
742 private void UpdateBookmarkPosition()
743 {
744     RectTransform bottomLeftPos =
            bookmarksScroll.GetComponent<RectTransform>();
745     Vector2 currentPosition = Input.mousePosition;
746
747     currentPosition.x -= bookmarkSize.x / 2;
748
749     float downVal = bookmarkMaskSize.y - (bookmarkSize.y / 2 *
            bookmarks.Count);
750
751     currentPosition.y -= Mathf.Clamp(downVal, bookmarkMaskSize.y / 2,
            bookmarkMaskSize.y);
752     bottomLeftPos.anchoredPosition = currentPosition; // Moves all
            bookmarks to the new position
753
754     Vector2 borderPosition = currentPosition;
```

```
755
756     borderPosition.y += Mathf.Clamp(0, downVal - bookmarks.Count *
757         bookmarkSize.y / 2, bookmarkMaskSize.y);
758     bookmarksBorder.anchoredPosition = borderPosition; // Moves the
759         bookmarks border to the new position
760 }
761
762 /// <summary>
763 /// Closes the currently opened menu.
764 /// </summary>
765 public void CloseMenu()
766 {
767     BehaviorManager.Instance.LockUI = false;
768     reopenBookmarks = false;
769     Invoke("UnlockUI", 0.1f);
770     background.SetActive(false); currentMenu.SetActive(false);
771
772     if (currentMenu == addMenu) addStartingPanel.anchoredPosition =
773         addCustomPanel.anchoredPosition = Vector2.zero;
774
775     currentMenu = null;
776     enabled = false;
777 }
778
779 /// <summary>
780 /// Allows the bookmarks menu to be opened; called by invokement
781     within this script.
782 /// </summary>
783 private void UnlockUI() { reopenBookmarks = true; }
784
785 /// <summary>
786 /// Creates a starting circuit from its index representation.
787 /// </summary>
788 /// <param name="startingCircuitIndex">Index of the starting
789     circuit.</param>
790 /// <returns>The newly created circuit.</returns>
791 private Circuit GetStartingCircuit(int startingCircuitIndex)
792 {
793     switch (startingCircuitIndex)
794     {
795         case 0:
796             return new InputGate();
797         case 1:
798             return new Display();
799         case 2:
800             return new Buffer();
801         case 3:
802             return new AndGate();
803         case 4:
```

```
799         return new NAndGate();
800     case 5:
801         return new NOrGate();
802     case 6:
803         return new NotGate();
804     case 7:
805         return new OrGate();
806     case 8:
807         return new XOrGate();
808     default:
809         throw new Exception("Invalid starting circuit index.");
810     }
811 }
812
813 /// <summary>
814 /// Serializes the current editor scene; called by pressing an in-  ↗
815   scene button.
816 /// </summary>
817 public void Serialize() { EditorStructureManager.Instance.Serialize  ↗
818   (); }
819
820 // Getter methods
821 public static TaskbarManager Instance { get { return instance; } }
822
823 public bool ReopenBookmarks { get { return reopenBookmarks; } }
824
825 public GameObject CurrentMenu { get { return currentMenu; } }
826
827 public List<int> BookmarkIDs { get { return bookmarkIDs; } }
828 }
```