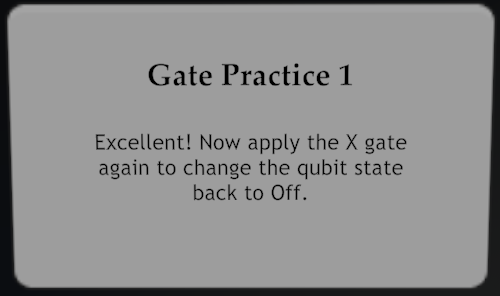
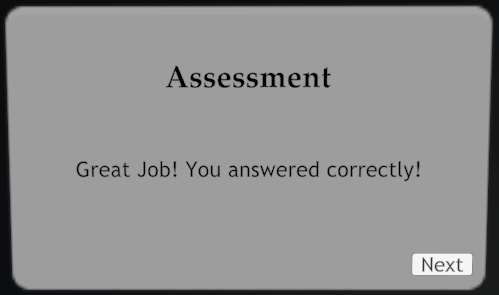
**Module Creation**

This section will detail how to create additional modules for QubitVR. Each module consists of a set of menus chained together, each one either a menu with a next button, or a menu without a next button. These are the only two types of menus that exist within a module.



GameObject Structure

A new scene is required for each new learning module. We recommend duplicating an already existing scene, and remaking the section manager game object. The structure of the section manager object is as follows:

**Manager** - An empty game object that acts as the parent for the Assessment and Tutorial game objects. This object should have the ‘ModuleManager’ script attached to it.

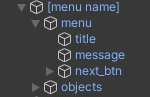
**Tutorial** - An empty game object that acts as the parent for all of the tutorial menus. This object should have the QubitManager and ‘Section # Tutorial Manager’ script attached to it.

**Assessment** - An empty game object that acts as the parent for all of the assessment menus. This object should have the QubitManager and ‘Section # Assessment Manager’ script attached to it.



**Next Button Menus**

Within the Tutorial and Assessment objects exist the menus that create your module. As previously mentioned, there are only two types of menus: those with a next button and those without a next button. This section will detail how to deal with menus with a next button.



Each menu is housed inside of an empty game object that is used to uniquely name each menu. The structure of this game object is as follows:

**[Menu Name]** - An empty game object that acts as the parent to contain ‘menu’ and ‘objects’

**Menu** - The parent menu object that contains the three main menu components: title, message, and next\_btn.

**Title** - The title text of the menu

**Message** - The body text of the menu

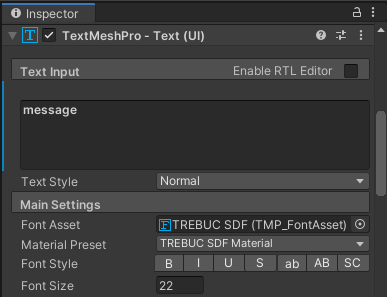
**Next\_Btn** - The next button in the lower right corner of the menu

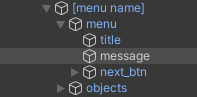
**Objects** - An empty game object used to contain any objects that you would like to show up with the menu. For example in the figure below, four gates appear with the menu.



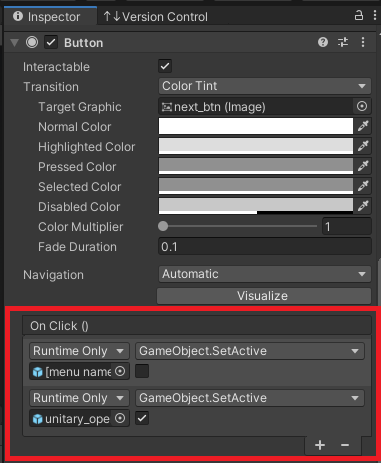
Customizing The Module

Changing the contents of the ‘Title’ and ‘Message’ fields are straightforward. After selecting whichever component you’d like to edit, it can be edited by using the TextMeshPro section in the inspector tab.



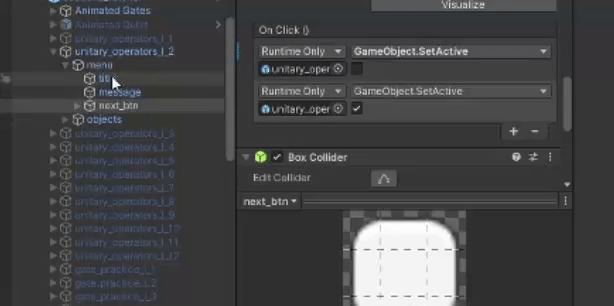


Editing the behavior of the ‘Next\_Btn’ field requires a more detailed explanation. After selecting the ‘Next\_Btn’ object in the scene hierarchy, the Button section will appear in the inspector tab. This is where we can add our onClick() functions in order to enable and disable menus and gameobjects using the next button.





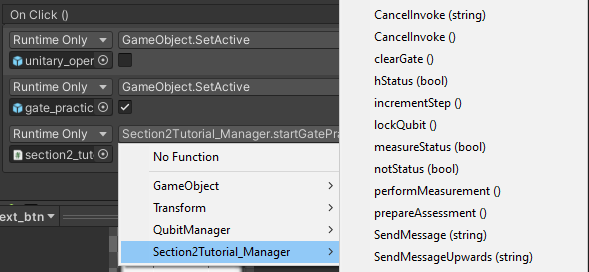
If the OnClick() field is empty and you want to add a behavior to it, click the ‘+’ button in the lower right section of the OnClick() section. Typical convention for these onClick() functions is to disable the current menu and then enable the next menu. This can be done by dragging the parent object [menu name] described above into the ‘Select Object’ slot.



After the desired object is placed into the ‘Select Object’ slot, use the right-most dropdown menu to find ‘GameObject.SetActive’. This allows you to set the boolean value to either ‘true’ or ‘false’ by toggling the checkbox.

Referencing Scripts With OnClick()

If you would like your ‘next button’ to call a function in a script, you need a way to reference that script in the dropdown menu in the OnClick() section. Recall that the parent object of the current section (either ‘Tutorial’ or ‘Assessment’) should contain a manager that controls this section. Attaching this parent game object will give you access to the script it contains along with all of the public functions within. This will be the first major use for the ‘Section # Tutorial Manager’ / ‘Section # Assessment Manager’ scripts.



By attaching the ‘section2\_tutorial’ gameobject to the ‘Select Object’ slot, we now have access to any scripts attached to it, and thus all of the functions within. This can be used for transitions to the assessment, practice sections, or any other functionality that cannot be covered by turning a menu on or off.

The ‘Section # Tutorial Manager’ / ‘Section # Assessment Manager’ scripts will be further utilized with menus without next buttons. The next section will cover that in more detail.

**No Next Button Menus**

Since there is no next button to click on these menus, we need an alternative method to detect user input to progress to the next menu / stage. This will be done using the Update() function in the ‘Section # Tutorial Manager’ / ‘Section # Assessment Manager’ that is attached to the current section.

The breakdown of variables and branches within the Update() function is listed below. The corresponding script in which to access these variables and methods will also be listed:

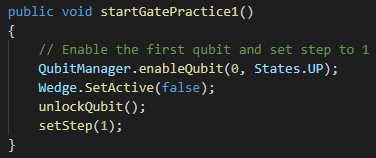
**State** - Variable used to lock the entire Update() function. True when you want the update function to be active, false when you want it to be inactive. This is useful for controlling when the Update() function is listening for certain conditions.

Accessing ‘State’ in code can be done by directly calling setStateTrue() or setStateFalse(). Both reside in ‘ModuleManager’ which should be inherited by your ‘Section # Tutorial Manager’ / ‘Section # Assessment Manager’.

Accessing ‘State’ in the OnClick() section can be done by clicking and dragging the parent ‘Section N Manager’ ***game object*** into the ‘Select Object’ slot. The ‘ModuleManager’ script should now be accessible for you to select either setStateTrue() or setStateFalse().

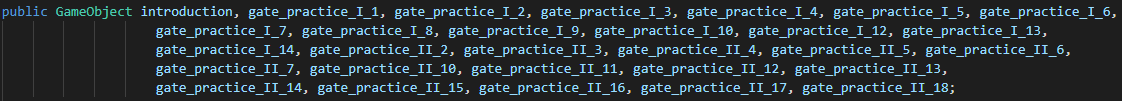
**Step** - Integer variable used to iterate through the switch statement. This determines which condition is being listed for whenever the user is on a menu with no ‘next button’. This is used by either calling ‘incrementStep()’ in code or by attaching the ‘Section N Manager’ ***game object*** and accessing it through the OnClick() section. ‘incrementStep()’ will increment the step value by 1.

It is good practice to manually set the step to 1 by using the ‘setStep(int n) function. This is usually done in a method that kick-starts the section and ensures that the step starts the section with a proper value.

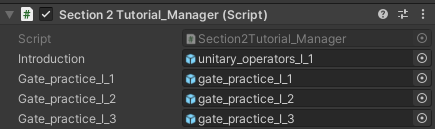


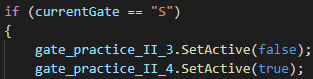
**Condition** - This is the condition to be listened for on a menu without a next button. This condition is completely in control of the developer. An example of a condition to listen for an S gate would be ‘if (currentGate == “S”)’ where currentGate is a reference to the qubitHistory that keeps track of the current gate that was just applied. To reiterate, the developer has complete creative control over ‘condition’.

**Menu** - This is a game object referencing the menu to be loaded / unloaded whenever ‘condition’ is met. The general rule is that all menus without a next button AND all menus that follow a menu without a next button need to be declared as public GameObjects at the top of the script.

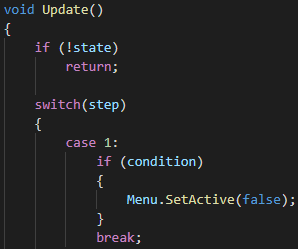


These game objects are to be populated in the Unity editor with their respective menu in the scene hierarchy. The typical procedure is to unload the current menu and load in the next menu whenever the condition is met.





This can be achieved by using the GameObject.SetActive() function. Pass in ‘true’ to enable the game object or pass in ‘false’ to disable the game object. An example of all of these components in action is provided below.



Using these tools, you should be able to fully create your own modules. Similar to the ‘condition’ statement, there will be other creative liberties that need to be taken in order to customize and tailor the module to your needs. Searching through the many scripts such as QubitManager and GateHistory should offer enough tools to get started.

**Scripts**

ModuleManager

Contains most general purpose functions for controlling components in a module. This function should always be inherited by the ‘Section # Tutorial Manager’ / ‘Section # Assessment Manager’ scripts. This function should always be attached to the ‘Section N Manager’ ***game object***.

Section # Tutorial Manager and Section # Assessment Manager

These scripts should always be attached to either the ‘Section # Tutorial’ or the ‘Section # Assessment’ gameobject respectively. These scripts should always inherit the ‘ModuleManager’ script. The main purpose of this script is to control what happens within the Tutorial or Assessment sections via the update() function and any other helper functions that go along with it.

QubitManager

When creating additional modules, the QubitManager script must be attached to both the Tutorial and Assessment game objects. The only field required for the QubitManager is the array of qubits. These are the qubits that will be displayed in the section that the QubitManager is attached to. After the ‘size’ parameter is set to the number of qubits in the scene, each element must be populated with the corresponding qubit game object that exists in the scene hierarchy.

