Working with Unity

Phaser Configurator

# Unity

Unity is an application framework mostly intended for the development of video games – it helps developers not need to worry about fetching input, rendering graphics, implementing physics, or managing all of the different segments of code that need to be run, letting them focus on just making the application. However, it can be easily repurposed to make a non-gaming application, such as the Phaser Configurator.

An application made in Unity is broken down at several levels – each application is made up of at least one Scene, which consists of at least one GameObject, which consists of at least one Component. Components could be considered *modifiers* to the GameObject *entities*, as each Component modifies the behavior of each GameObject – some Components add colliders, some add gravity, some allow the GameObject to have an appearance, and some render other GameObjects to the screen. Developer-made scripts are also considered Components (so long as they extend from *MonoBehaviour*), which is how logic specific to each application is implemented.

# Required Tools

## Unity Editor

To do any sort of modification on the project, you will need to install a copy of the Unity Editor, the application used to produce anything built on the Unity framework. A copy of the Editor can be fetched from [this webpage](http://unity3d.com/get-unity/download?ref=professional).[[1]](#footnote-1) While recommended to install all three segments of the Editor, installation of the Standard Assets and the Development Web Player are not completely necessary.

After installing the Editor, activating the Pro License for use should be done. Under the Help menu up top should be the “Manage License…” option. Select that, then choose “Activate New License” and follow the prompts that show up.

## A C#-Capable IDE

To be able to modify the code, you would want to find yourself an IDE that’s capable of working with C#. Visual Studio would work best, however the license on the free version doesn’t allow organizations like Star to use it and the Professional version costs approximately US$1200. Meanwhile, Unity comes with MonoDevelop, which is sufficient and free but not as polished as Visual Studio.

## Git

Git is a version control solution that allows more than one person to work on a project simultaneously, as well as backing up project assets to be able to revert changes made to a specific segment of code. I would suggest that you use the [GitHub for Windows](https://windows.github.com/), however MonoDevelop and newer versions of Visual Studio has Git control built-in as well.

While GitHub for Windows might request you link the application with a GitHub account, it’s not necessary as the only repository you will need to interact with is hosted on the U:\ Drive. You can “Skip setup” on the first screen, though I would suggest you go into GitHub for Windows’s Options (via the gear under the Close button) and switch your default shell (on the right) to Git Bash. If desired, you can also configure Git (to the left) so your name and email is attached to every change you make to the project.

# Getting the Project

As of writing, the project resides on my Personal Folder, at [1000LightbarConfig2](file:///U:\Personal_Folders\Christopher%20C\1000LightbarConfig2). How you fetch the project from that location will depend on what tool for Git you’re using.

## Via GitHub for Windows

Installing the GitHub for Windows app will have also added a shortcut to the Git Shell to your Start Menu. That shortcut is a quick link to whatever preferred shell you’ve selected in the application.

To copy down the project, open the shell, and use cd to navigate to where you will want the project to reside. If you’re using Git Bash (as recommended above) you can navigate to the U:\ Drive using *cd /u*, else you will want to use just *U:*. Once you’re in a suitable location, copy the project there via:

git clone <repo location> <folder name>

…where “repo location” is the location where the repo is residing (as of writing, that would be “*/u/Personal\_Folders/Christopher\ C/1000LightbarConfig2*”), and “folder name” is the name you want to give your local copy of the project (which does not need to be “1000LightbarConfig2” if not desired – if the folder name is omitted, it will name itself 1000LightbarConfig2).

Once complete, you may close the shell (either by issuing the exit command or using the close button on the window). Return to the GitHub for Windows window, and click the plus on the top left. Choose to *Add* the local copy of the project by having it point to the folder you just created.

## Via Visual Studio

Once Visual Studio is open, use the View Menu up top to access the Team Explorer pane (second item). Click Connect to Team Projects (looks like a plug: ) and the second section of that pane should be a listing of local Git repositories. Click Clone, then enter the URL of the Repo in the upper box (as of writing, that would be “*U:/Personal\_Folders/Christopher C/1000LightbarConfig2*”) and wherever you’d like to store the repository in the lower box. Then hit the Clone button.

## Via MonoDevelop

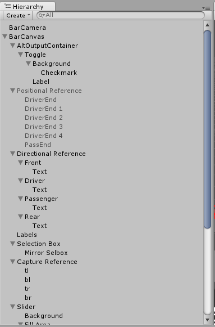
Once MonoDevelop is open, use the Version Control Menu up top to access the Checkout utility (first item). Change the Protocol to “file” and enter the URL of the Repo into the Path box (as of writing, that would be “*U:/Personal\_Folders/Christopher C/1000LightbarConfig2*”). At the bottom, change the Target Directory to wherever you want your copy of the project to reside, and hit OK.

# Opening the Project

First, open the Editor. This will be the application that has the default Unity cube icon, and is labelled as “Unity” – this might appear on your Desktop, otherwise it will be in your Start Menu. For the first time opening the project, use the white “Open other” button at the top and navigate to wherever you’ve saved the project, choosing the folder you’ve copied down. Afterward, Unity will save the project in a “Recent” list that will show on every subsequent opening of the Editor.

# The Editor

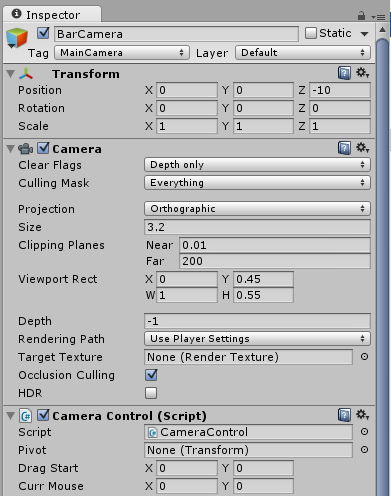
## The Hierarchy

The Unity Editor utilizes a variety of tabs to display various groups of important information about the project. For instance, the listing of each GameObject in the Scene can be found in the Hierarchy view as depicted on the right. Each GameObject may consider another GameObject it’s “parent,” meaning that its own position in the 3D space is dependant on another’s. Additionally, each GameObject can be enabled or disabled – only enabled GameObjects are able to function as their Components describe.

A *disabled* GameObject will appear slightly gray on the Hierarchy, to indicate that it still exists even if it’s not doing anything. The Hierarchy also allows you to collapse children, hiding them from view to help organize your GameObjects.

Speaking specifically in regards to the Phaser Configurator, the application can get away with only one Scene (the other Scene in the project is a simple splash that doesn’t do very much). Additionally, there are two “Camera” GameObjects, one to render the in-progress light bar and one to render the GUI – the latter sits below the former. There are also two different Canvas GameObjects that define where GUI elements may be displayed – one handles the labels on the light bar itself while the other one handles the overall GUI (light bar editing, order completion, light pattern editing, etc).

## The Inspector

In order to tweak a selected GameObject to modify its behavior, you use the Inspector pane (as depicted on left) – with it you can change its Transform (it’s position, orientation, and size in the 3D world), modify any of the Components attached to it (such as how the Camera renders or which functions are considered “steady burn”), or even rename it. If you want to select something else but still view this GameObject’s Inspector values, click the padlock icon above the top right corner of the pane.

As far as logic goes, every scrap of code specifically for the Configurator is written in C#. If you don’t have an IDE like Visual Studio that can handle it already (Notepad++ doesn’t really count) Unity comes with an IDE called MonoDevelop built specifically for working with Unity. In case you need it, every Unity-related class’s documentation can be found at <http://docs.unity3d.com/ScriptReference/>.

## The Project View

The Project View serves as a file browser / asset browser for every asset that exists for the project. All assets can be organized into a folders like in Windows Explorer. Simply double-click a folder to navigate into it, and double-click a file to open it up.

Selected assets can also be modified – by selecting an asset, the Inspector View will change to display information on that asset. This can be used to change prefab script variables or change how a certain image is imported.

This project contains three Scene files (which bear Unity’s icon), even though only two of them are actually in use. “splash” is the Scene that is loaded immediately upon opening, and will do nothing except display a pretty graphic as the application loads the next Scene, “scene”, which is where all of the magic happens. “orthoSizeFinder” was a Scene created to calculate out what the necessary camera sizes were for bar image capture.

The project also contains Prefabs (which bear a blue cube icon) – Unity’s equivalent of Star’s phantom assemblies. They hold information that Unity can use to create quick clones of a GameObject and any of its ancestors. Useful when you have a lot of a certain GameObject which change in number frequently (such as the variety of lists).

## The Scene View

When you need to manipulate the position of something (including GUI elements, the various cameras, light heads, and a few other miscellaneous items), the use of the Scene view gives you a free-roaming camera with which you can view how objects are positioned in the 3D space that Unity lives in. Various objects will render themselves in the Scene View that won’t show up in the application itself, such as the red lines that show just how the ReferencePoints will line up, or the black trapezoids that represent the light heads, or the big D that indicates where the LightDict GameObject is.

You can select GameObjects to manipulate either via the Scene View or via the Hierarchy via the Left Mouse Button. Click-and-drag to box select multiple GameObjects will work on the Scene View.

You can rotate your view in a first-person schema by clicking and dragging the Scene View with your Right Mouse Button – continue holding the button and use the WASD keys to fly the camera’s position around the world (Q and E will move you vertically down and up, respectively, as well).

Alternatively, you can opt to use the Alt key to move yourself around the world. Hold Alt and click and drag with the Left Mouse Button to orbit. Alt and dragging the Right Mouse Button can be used to zoom. Dragging with the Middle Mouse Button (with or without Alt) is used to pan around on the same plane as your current view.

Finally, if you can’t find a certain GameObject you’ve selected, hold the mouse over the Scene View and press the F key, and the Scene camera will immediately frame it for you, centering the object in your view and zooming out enough to put the entire extents of it in view.

## The Game View

The Game View can be used to visualize the application as it would appear to the end user once completely built. Additionally, this view is used to test the application without building it – simply use the Play button at the top of the Editor to begin the application, and everything in the Game View becomes as live as a built version.

At the top left of the view is a dropdown that you can use to restrict the resolution of the application – useful when testing it against a 1024 x 768 resolution (what I’m calling the minimum for the application). You can select either a specific aspect ratio or a specific fixed resolution to test against.

On the top right are a few controls that should be handy. “Maximize on Play” should be self-explanatory, as should “Mute audio” (though the app doesn’t use any sort of sound). “Stats”, when enabled, will list off various performance statistics that might be handy to keep tabs on – screen resolution, FPS count, how long a frame is taking to process, and some graphical statistics. “Gizmos” will turn on any visualization that’s seen on the Scene view, the drop down lets you select exactly what gizmos you want to see.

A word of caution: while the code can be edited “live” – the code can be modified while the Editor is still previewing the application – it will start throwing many errors as it reloads all of the scripts, causing the light heads to lose their definitions and the Configurator to start throwing fits from the null references.

# The Code

As of writing this documentation, a majority of the scripts are meant to have only one instance in the Scene, including *CameraControl*, *BarManager*, and a few others. Again, because these scripts extend from *MonoBehaviour* they are considered Components to be added to GameObjects in the Scene. Most of the documentation you’ll need can be found in-line by simply hovering over the item you have questions about, however there are some key classes worth mentioning here.

Opening any of the code files can be done by simply double-clicking the code file out of the Project view.

## CameraControl

CameraControl is a Component that sits on the same GameObject as the Camera Component that renders the bar. It’s responsible for giving the rest of the application easy access to that Camera Component, as well as handling the selection of the heads and lenses. Also, this Component handles the callbacks for what should be refreshed every time a selection happens – these should all be assigned via Inspector. Lastly, this Component handles debug input (*RCtrl+RShift+[* to show the debug bits, for instance), the input for the scrollwheel and right mouse button to move the camera itself, and continually refreshes the resolution so Unity doesn’t glitch out with that one issue we had previously.[[2]](#footnote-2)

To learn a bit more about how to control the camera, I heavily suggest you check Unity’s documentation on the Camera component, as is available [here](http://docs.unity3d.com/ScriptReference/Camera.html). Every variable you would need to manipulate is available there.

## BarManager

The BarManager Component does exactly as its name implies – it manages the entire bar, from holding a list of every head that’s on it (visible or not), making sure each head gets the right bits, applying auto-phase on demand, holding a list of all of the lenses, tracking size, tracking pattern setup, and tracking accessories. However, it also manages a few other things, such as bar saving / loading, handling the quit callback (preventing the application from closing when the “close” button is pressed under certain conditions), and handling PDF production.

## LightDict

The LightDict Component manages references to everything that a light bar would require – light head optics and styles, lens options, light flashing patterns, and accessory options. This is the one Component responsible for loading information off of the provided *lib.nbt* file to populate the application with every mite of information it needs. Please see the other Word document I’ve written – linked on the right – to get a better understanding on how to mess around with the library file. If you add something completely new to the library file, you’ll want to modify the LightDict class to be able to handle it.

LightDict’s file also contains definitions for the various structures and enumerations that the project deals with – Advanced Functions, Basic Functions, Locations, Patterns, and the various options available. It also contains the Extensions static class, which defines a few helper methods I’ve used to perform a number of different tasks.

## LightHead

LightHead is a Component that exists on every light head on the bar. While it manages information regarding the physical definition of the head (optic and style) it will also cache information regarding patterns and enables, and handles default optics and styles when adding functions.

Immediately after adding a function, two things occur: first, the head checks itself so that the OpticSelect GameObject knows whether a single or a dual color can be applied. Second, it checks itself again to apply default optics and styles where possible – if only one style is “recommended” after checking all options against the function list, then that style is selected, otherwise it only picks the optic out for you. The latter processing can be skipped (say, when applying a savefile), allowing only the test for single/dual to happen.

## ErrorText

The ErrorText Component resides on a Text GameObject at the bottom of the UI. It’s useful in giving information to the user for a brief moment – whether it’s letting them know an error occurred (ie the PDF they’re trying to export to can’t be written) or informing them of some background tasks occurring (ie an error log being sent).

If you wish to make use of this, simply use ErrorText.inst.DispError(string) or ErrorText.inst.DispInfo(string) to display an error or display information respectively.

## ErrorLogging

While we still want to collect reported issues with the new Phaser Configurator, the ErrorLogging Component will assist in sending emails to all involved. When the user attempts to report an issue, the Component will capture a screenshot (if included) and send the emails via a pre-configured SMTP server with pre-configured credentials. Additionally, the ErrorLogging Component will handle logging input if the user enables it, capturing whatever mouse input the user performs. The Component will save the information into an internal 4 MB buffer (4194304 characters of space), and will send the contents of that buffer with the log if the user believes it would be helpful.

The GameObject containing this Component is the very last item in the Hierarchy, called “ErrorLogger.” Inspecting this Component on that GameObject will let you easily configure which email addresses receive the messages.

1. If a download isn’t possible (as Star’s firewall has a tendency to kill connections transferring more than 20 MB), a copy of the installation is available [on the U:\ Drive](file:///U:\Personal_Folders\James\Unity%205.1.1f1%20Install). Freshness not guaranteed. [↑](#footnote-ref-1)
2. …where Unity wouldn’t accept input for any “new” area after making the window larger. [↑](#footnote-ref-2)