

# Block Light 2D

Robronic Games



## Table of Contents

<b>Important</b>	<b>2</b>
<b>Instructions</b>	<b>3</b>
<b>How to implement in your game</b>	<b>7</b>
<b>Adding blocks to the Place Tile tool</b>	<b>8</b>
<b>Removing blocks from the Place Tile tool</b>	<b>8</b>
<b>Lighting system – How it works summary</b>	<b>9</b>
<b>F.A.Q.</b>	<b>10</b>
<b>Extra details</b>	<b>11</b>

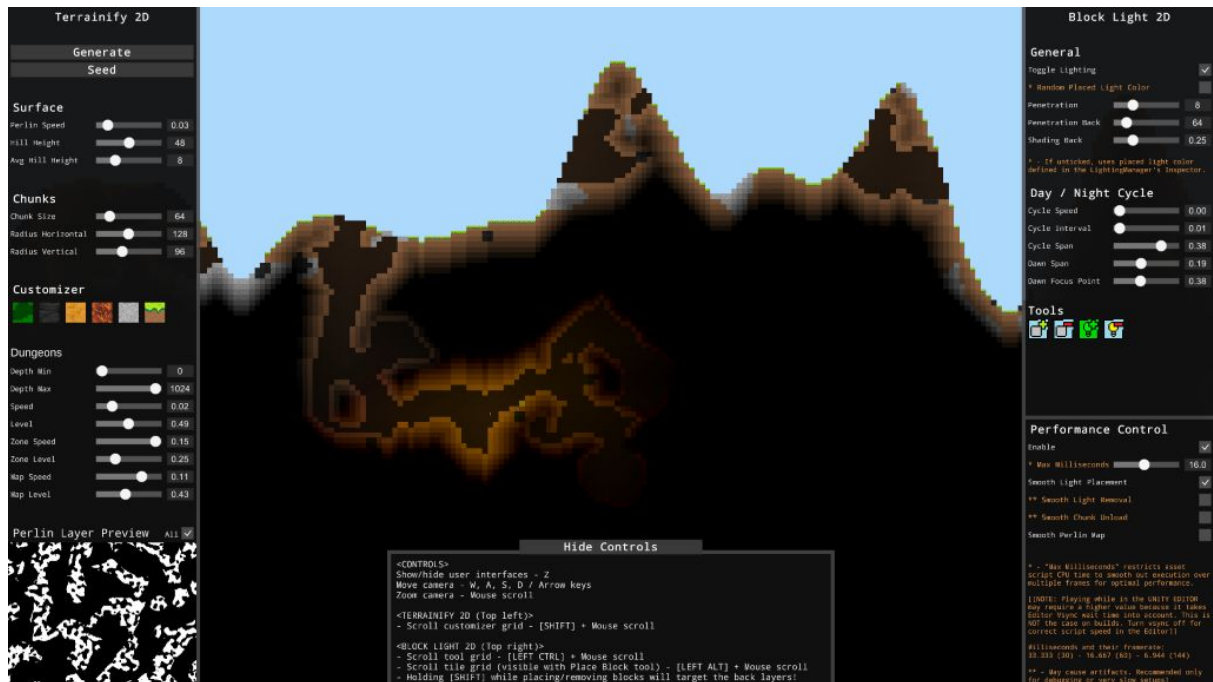
## Important

This guide will always detail the latest version of this asset and I intend to cover as much as possible to give you clarity on what this asset can do, how you can use it, how to port it over to your own games et cetera.

This asset is a combination of two since version 1.1, consisting of **Block Light 2D** which is the core of this asset and **Terrainify 2D**, my free asset as support that gives you some simple terrain generation combined with a robust chunk loading system. **Anything related to Terrainify 2D can be found in its own guide included in this package. This guide will only cover the functionality of Block Light 2D.**

## Instructions

This asset comes with a demo scene that includes a simple interface with tools for you to experiment with the lighting.




## CONTROLS


All controls can be seen in the Controls window at the bottom of the screen.

The top (right) section features the following sliders and settings:


## GENERAL

General controls and core lighting settings

- **Toggle Lighting** ☒ - Toggles all lighting in the game on or off. This only affects the front end visuals (the Tilemaps). Light is still being calculated and stored in the back end. It's initially designed to make it easier for you to experiment with the generation options **Terrainify 2D** has to offer;
- **\* Random Placed Light Color** ☒   
 \* - If unticked, uses placed light color defined in the LightingManager's Inspector. - Toggles whether to use a randomly colored light or a light with a color defined in the LightingManager's exposed light color field in the inspector when using the  Place Light tool.
- **Penetration**  - Controls how deep the light can penetrate the **front** Tilemap layer, as simple as that. Default value is 8;



-  - Controls how deep the light can penetrate the **back** Tilemap layer. I found a good balance for this to be about 8x the front Penetration. Default value is 64.

Its max value is 1024 in case you want to test deep terrain penetration, as seen in the 1.1 update video. You probably want to tweak the terrain generation to generate more holes though to fully utilize depth penetration;


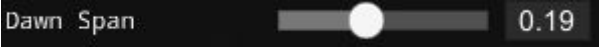

-  - Controls the darkening of the back layer compared to the front layer. This is a simple multiplication factor with the color to determine the final color of the back layer. This means that low values will result in a very dark back layer, high values make it appear brighter. Default value is 0.25.

## DAY / NIGHT CYCLE

Core controls and settings for the day/night cycle system

-  - The speed of the cycle. Realistically you won't need a high cycle speed, I found that anywhere from 0.01 to 0.1 already gives great results.
-  - The time (in seconds) between visual updates of the cycle. The slower your cycle speed, the higher you can set this interval before the change becomes too great and can be noticed. You probably want to find a good balance between the speed and this interval depending on your game.

NOTE: Even though the cycle is regulated by the performance control system, it can result in high CPU performance with low values.

-  - This controls the range of darkness the cycle causes. For example the max value of 0.5 will cause the darkest point of the cycle to set the tiles completely black. 0.25 would be 50% brightness and so on. Base value is 0.38 which'll darken the scene well, but still allows some light to penetrate the ground.
-  - This controls the range of dawn over the whole cycle. Lower values will have a very short dawn period while greater values stretch the dawn time. I generally found that making this half the Cycle Span is a good starting point.
-  - This controls the point of the cycle where dawn is situated. Just like Cycle Span it intends to work as an offset. Standard value is 0.38.


NOTE: Day/night cycle colors are defined in the **LightingManager's** inspector as exposed **Color** fields. They can be tweaked there!

## PERFORMANCE CONTROL

System that monitors and restricts asset script CPU times to smooth out execution over multiple frames for optimal performance.

-  - Enables performance management.

Recommended to always be on.

- 

\* - "Max Milliseconds" restricts asset script CPU time to smooth out execution over multiple frames for optimal performance.

[[NOTE: Playing while in the UNITY EDITOR may require a higher value because it takes Editor Vsync wait time into account. This is NOT the case on builds. Turn vsync off for correct script speed in the Editor]]

Milliseconds and their framerate:  
33.333 (30) - 16.667 (60) - 6.944 (144)

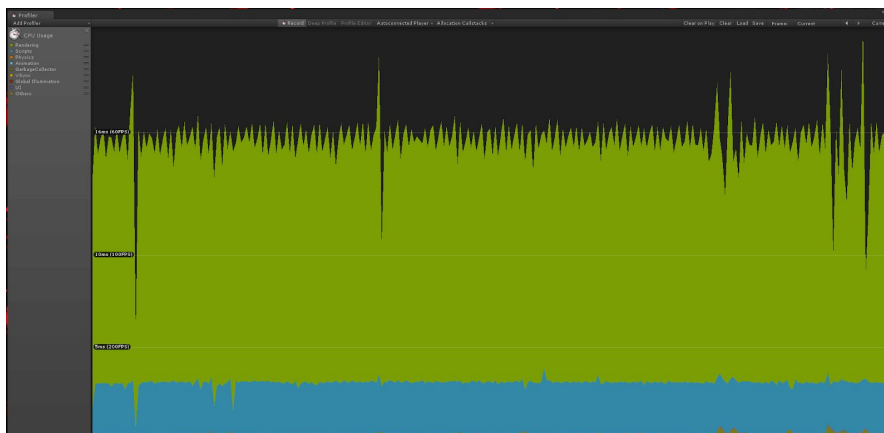
- Pretty self explanatory. Works for

Coroutine operations only and a script doesn't necessarily needs to be part of the asset. You can hook your own scripts into this as well. Simply request `HitFrameLimit()` to see whether there's processing time left for the current frame and act accordingly. Example of the day/night cycle update:

```
if (PerformanceManager.Instance.HitFrameLimit())
    yield return null;
```





**Like it says, if you want correct Editor script speed turn off Vsync under Edit - Project Settings - Quality and setting "V Sync Count" to "Don't Sync". A build doesn't suffer from this so if you make a build, I recommend to turn on Vsync again to prevent screen tearing.**

With Max Milliseconds set to for example 3 ms, the whole system will only use a maximum of 3 ms so you have full control over the performance. Profiler snapshot of a build with Max Milliseconds set to 3 ms and extreme cycle settings for stress testing:



**NOTE:** If you're not satisfied with the min and max slider values, you can adjust them yourself in their respective UI of course. If you want to tweak the slider values manually (not recommended), you can find their data storage under the **Scriptable Objects** folder as **LightingData** together with **CycleData** and **PerformanceData**, or **ChunkData** and **SurfaceData** for **Terrainify 2D**. **Terrainify 2D's Customizer** slider values are stored in the same folder under the **Customizer** subfolder.

Below the top section are a few basic tools I created to make it easier to experiment with the engine. Navigation through these tools is **detailed in the Controls window at the bottom of the screen** or simply clicking the tool you want to use. Some tools show a ghost and only when the ghost is green, the tool can be used on that spot. This system utilizes the **ToolManager**, which is by default incorporated under **UserInterfaceLighting**. The **ToolManager** is only meant to be part of **UserInterfaceLighting** and is **not** standalone!

-  **Place Tile** – When this tool is selected, the Tile Selection Grid appears at the bottom where you can choose the type of tile to place. To place a tile, use left click (or hold and drag).
-  **Remove Tile** – To remove a tile, use left click (or hold and drag).
-  **Place Light** – To place a light, use left click (or hold and drag).
-  **Remove Light** – To remove a light, hover over a light and use left click (or hold and drag).

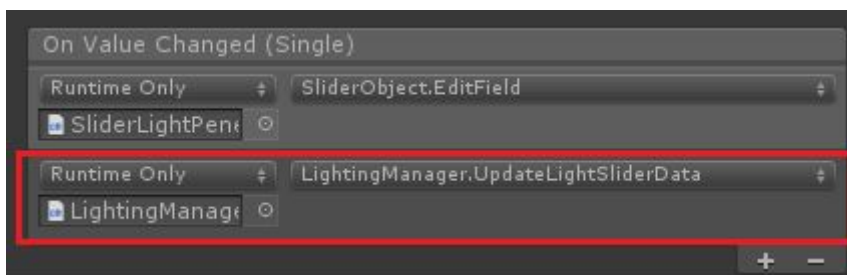
**NOTE:** Left click targets the front layer for placing/removing blocks. To target the back layer instead, hold **Left SHIFT**. It's detailed in the controls panel too.

## How to implement in your game

The user interfaces are standalone and can be disabled if necessary. They are only there to help you change all settings in a structured manner. There are only at most 3 things you need in a scene to make this asset work completely:

- A **Camera** in the scene;
- A **[Block Light 2D]** GameObject. Simply drag and drop the **[Block Light 2D]** prefab into your scene;
- **(Optional)** If camera movement and zoom are needed, add **SimpleCameraBehaviour** script as a component to the scene's **Camera**. If you want a “fancy” night sky background, drag and drop the **Background** prefab into your scene.

**NOTE:** Right now all interactable UI elements use the **UnityEvents** exposed in the editor to notify core objects of changes. For example a lighting slider notifies the **LightingManager**:



This means that if you delete a UI object from the **[Block Light 2D]** object and later decide to drag a new UI object in the scene, these references will not be there. That's why they're encapsulated into the **[Block Light 2D]** prefab. **If you don't need a UI, simply disable its root GameObject or, as the control window says, press [Z] to hide these UI's.**

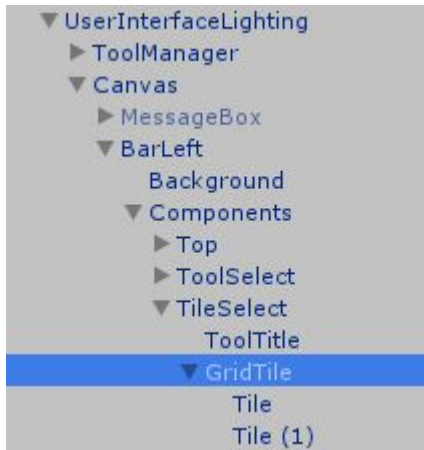
The **Terrainify 2D** guide explains how to add collision detection to chunks to have a player walk over the world etcetera, also under the how to implement section. **Do note that these are general directions and I haven't fully tested it since it's not an integral or required part of this asset.**

**IMPORTANT:** The user interfaces work with an **EventSystem** object. It is automatically added to the hierarchy whenever you create a **Canvas** for example. **If your game doesn't have one and you drag a UI into the scene and play or during play, you'll get NullReference errors.** Simply go to **Create – UI – Event System** from the scene hierarchy to add one if there is none, you don't have to change anything to it. It's added by default to the **[Block Light 2D]** prefab.



## Adding blocks to the Place Tile tool

If you happen to really want or need more blocks for the Place Tile tool, you simply go to the Tiles grid in **UserInterfaceLighting**:



Then select for example Tile (1) and hit CTRL+D to duplicate it. Then you only need to change two things in the new object:

- Change the **Source Image** in the **Image** component to the desired tile, for example **Coal**;
- Scroll down to the **TileItem** component and change the Tile dropdown menu to the block you wanted to add, for example **COAL**.

The grid in-game now has the block and you can choose it through navigation or clicking.

**NOTE: In case you want to add a new tile type, see the guide for Terrainify 2D!**

## Removing blocks from the Place Tile tool

Simply delete the GameObject under **GridTile** and it'll disappear.

## Lighting system – How it works summary

This engine uses a BFS (Breadth-First Search) algorithm by using a FIFO (First In, First Out) queue system. A quick example of how it works is the following pseudo-code for

### **UpdateLight:**

- Create a **LightNode** at a light's position and add it to the update queue
- While the update queue is not empty
  - Take the front **LightNode** out of the update queue
  - Look at its left neighbor. If its light value is quite lower than his, adjust its color to his color minus block falloff and add a new **LightNode** for that tile to the update queue.
  - -Repeat for down, right and up direction-

This ensures every tile or block is only visited once and results in fast computation.

**RemoveLight** is a little more complicated, but it's quite similar:

- Create a **LightNode** at a light's position and add it to the removal queue
- Clear the tile at the initial light's position to black
- While the removal queue is not empty
  - Take the front **LightNode** out of the removal queue
  - Look at its left neighbor. If its light value is lower than his, clear its color to black and add a new **LightNode** for that tile to the removal queue. Else add a new **LightNode** for that tile to the **update queue**.
  - -Repeat for down, right and up direction-
- Run the **UpdateLight** code from the while-loop

We run the while-loop part of **UpdateLight** again to fill in the blanks by spreading the light from brighter tiles I came across while removing light. This means that the light from other nearby light sources that intersected with the light we tried to remove, will fill in the blank space our removal left behind.

More details can be found in the code itself.

## F.A.Q.

**Hey, I get a NullReference error whenever I try to add a `UserInterfaceLighting` or `UserInterfaceTerrainify` to my scene. Help! What do I do?**

Go to **How to implement in your game** and read the final paragraph, it is detailed there.

**Hey, I get an error when dragging and dropping any UI prefab into my scene “Assertion failed on expression 'modifications.empty()'”! What do I do?**

This bug is common and many people experience issues with it. It seems to be linked to GUI components on prefabs, such as a Canvas. The bug does not break the UI's or the asset in any way, so I'd say just look the other way for now. It's related to dropping it into the scene at runtime so simply having it in scene and activating it when you need it avoids it.

**It doesn't save my slider settings in build mode! Help! What do I do?**

This asset works with Scriptable Objects that store the slider data. Changes made to them will be saved only while playing in the **Unity Editor**. It is of course possible to make this save in build mode too, but is currently unsupported. You could use **PlayerPrefs** to store the data if you really want to export it or resort to **JSON** for example, but this asset was designed to work together with the Unity Editor.

**This is a really cool asset! How can I support you?**

You already have by buying this asset! **If you want to show more support, please feel free to rate my asset well on the Unity Asset Store. I'd appreciate this very much!** If you have suggestions, feedback or any kind of problem you can contact me by mail at <http://robronic.com/contact/> and I'll help in any way that I can. Thanks for buying my asset and for reading this!

## Extra details

This asset is part of an ongoing series of assets towards making a full 2D world. These assets are split off from my own game as I'm creating it. The total list of current and future assets:

- **Block Light 2D** (released, this asset);
- [Terrainify 2D](#) (released, incorporated into this asset) - A free and dynamic seed-based generator of 2D terrain using **Perlin** noise and Unity's **Tilemap** system, combined with a robust chunking system.
- **Block Water 2D** (to be announced) – A dynamic water system with Unity's **Tilemap** system.
- **[?]** – (Other asset components I plan to develop in this series)
- **World Engine 2D** (to be announced) – Incorporates all assets in this series and allows you to build your own 2D world right from the get go.