

# Block Lighting Engine 2D

*Robronic Games*







## Instructions

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

At the top left, there are three buttons:

- **Clear** - Clears all tiles, lights and internal generation data;
- **Generate** – Runs Clear and generates a simple terrain layout to test lighting with;
- **Seed** - Set a new random seed for the Generate button;

Below the buttons are a few basic tools I created to make it easier to experiment with the engine. Navigation through these tools is as easy as using the mouse scroll wheel 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.

-  **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. If (left) shift is held you can mouse scroll through the tile selector. To place a tile on the front layer, use left click (or hold). To place a tile on the back layer, hold (left) shift and left click (or hold).
-  **Remove Tile** – To remove a tile from the front layer, use left click (or hold). To remove a tile from the back layer, hold (left) shift and left click (or hold).
-  **Place Light** – To place a light, use left click (or hold).
-  **Remove Light** – To remove a light, hover over a light and use left click (or hold).

## Further Details

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
  - o Take the front **LightNode** out of the update queue
  - o 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.
  - o *-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
- While the removal queue is not empty
  - o Take the front **LightNode** out of the removal queue
  - o 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**.
  - o *-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. Still have questions? Feel free to contact me at <http://robronic.com/contact/> and I'll help in any way that I can. Don't hesitate to contact me too if you found a bug, you have tips on improving the code or if you have suggestions for features you'd like to see in this asset.