EasyColorize2D Documentation

version 1.0

Table of content

Introduction

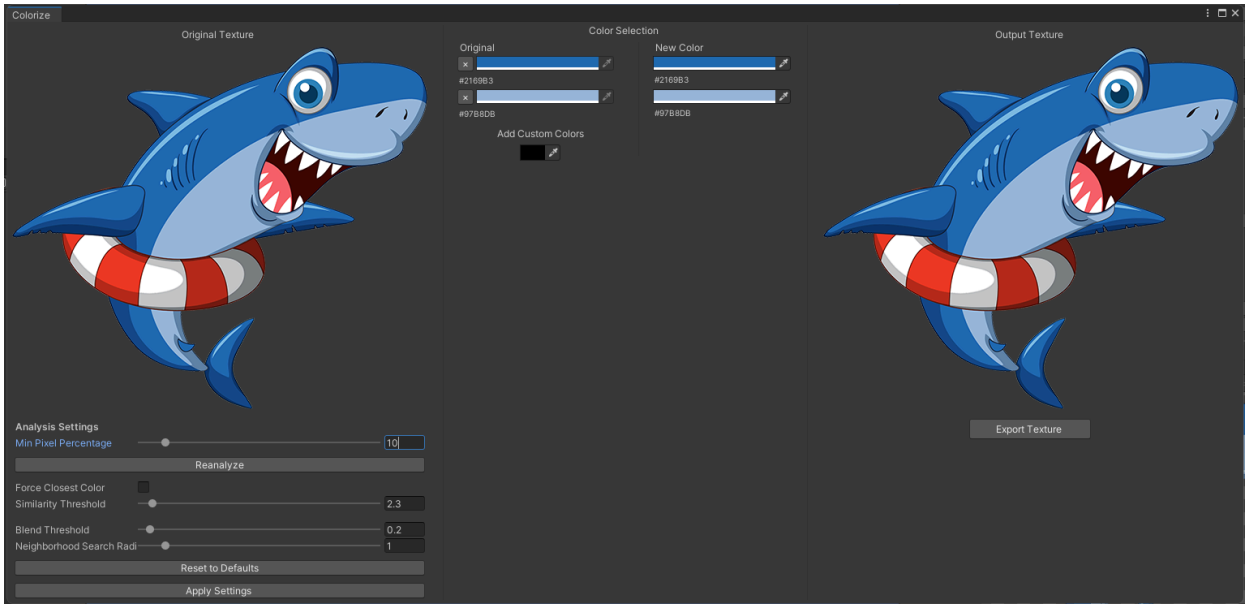
- **EasyColorize2D** is a powerful editor tool that simplifies sprite recoloring directly within your game development workflow. Unlike complex software like Adobe Photoshop, which requires extensive learning, **EasyColorize2D** allows you to swap colors instantly within a single editor window. With just a few clicks, you can select the color you want to change, choose a new color, and apply the transformation effortlessly.

Video Guides and Tutorials

- Intro & Full Tutorial
- Sample 2
- Sample 3

Where to Begin

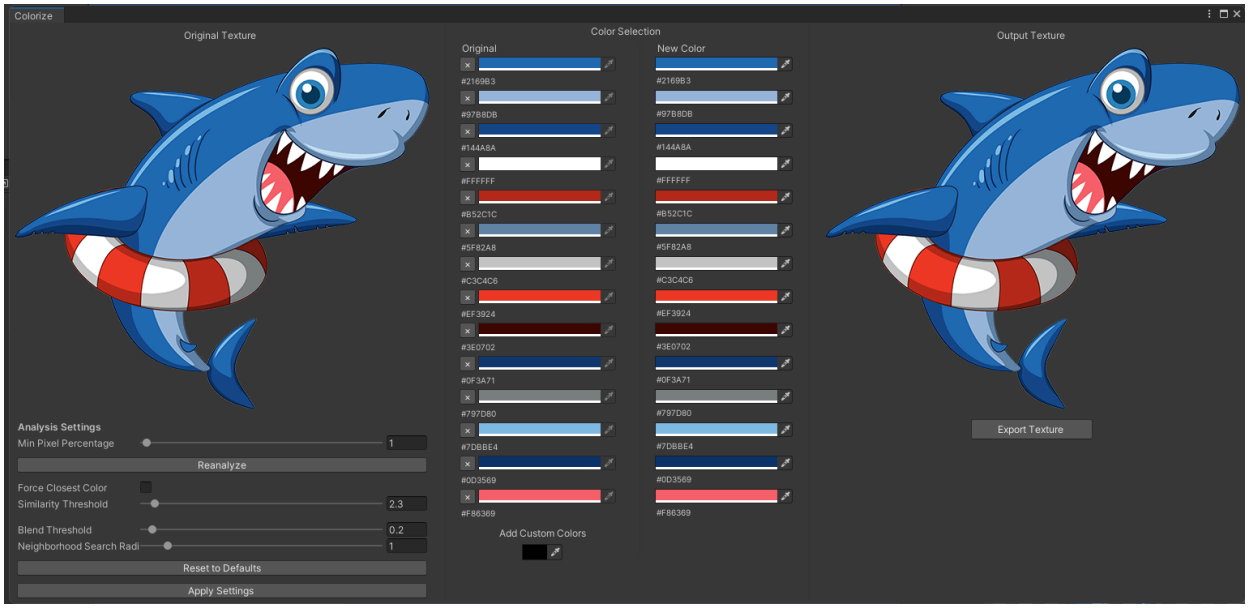
- Ensure that the **Read/Write** property of the texture is enabled and the TextureType property of the texture is **Sprite (2D and UI)**.
- In the Unity Editor, Right-click on the Image file you want to colorize in the Project view, then select **EasyColorize2D > Colorize**. The Colorize window will open which looks like the image below.



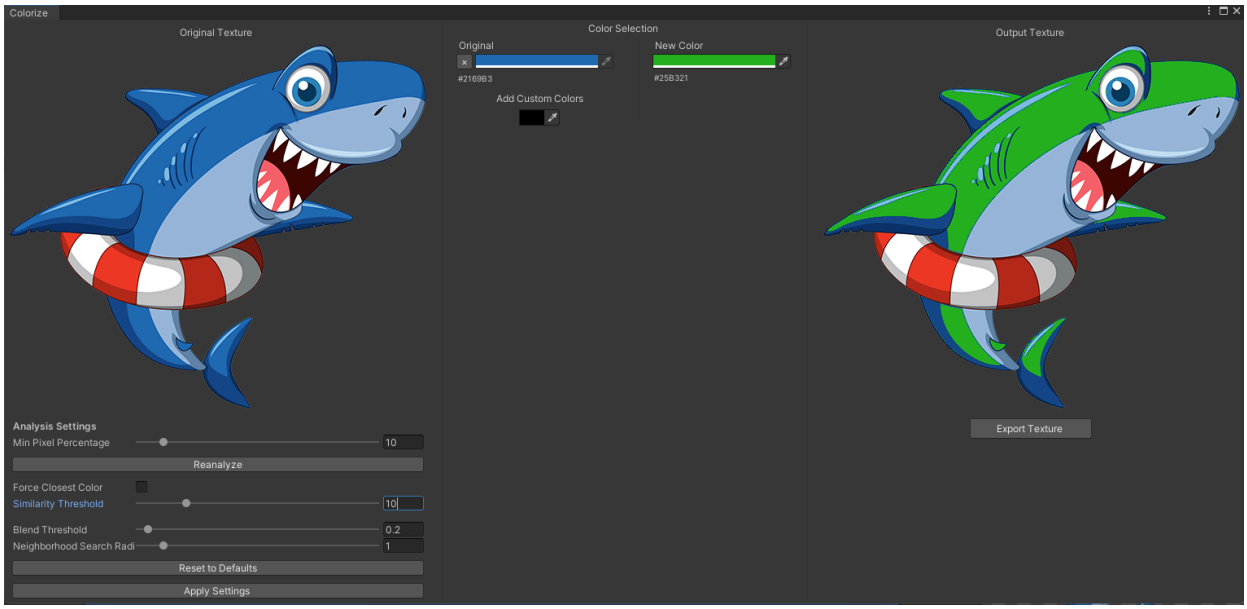
Colorize Window

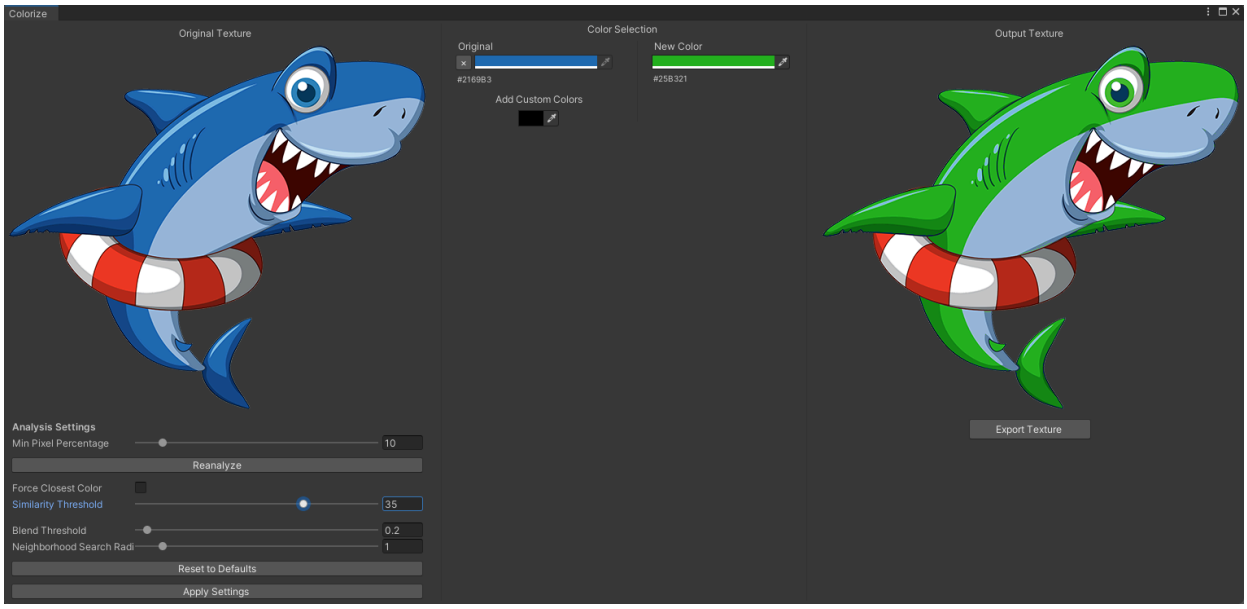
- This window provides everything you need to change the color of your sprites. I've also explained its functionality in detail in the YouTube videos with detailed examples, but let's break it down here as well.

- On the left, you'll see the original texture, and on the right, the texture after applying color changes. Initially, both textures are identical.
- In the center, there's a **Color Selection** section. The left column displays the dominant colors found in the original texture, which are generated through an analysis of the image.
- Below the original texture, you'll find the **Min Pixel Percentage** setting. Adjusting this value and clicking the **Reanalyze** button updates the dominant colors list, showing only those that appear in more than the specified percentage of non-transparent pixels.
- You can remove any of these colors or add custom ones manually using the color picker below the column.
    - For example, the image below shows the detected colors for the same sprite as above when **Min Pixel Percentage** is set to 1% instead of 10%.
    - Check out Rules of Thumb #1 and #2 for guidance on effectively using this setting.

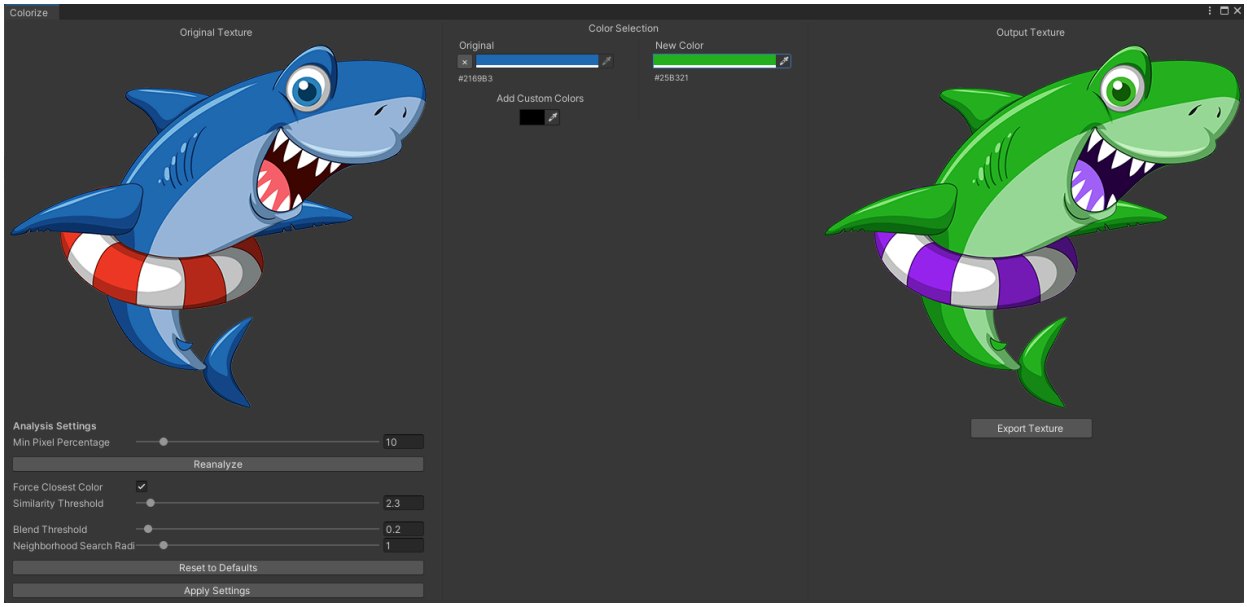


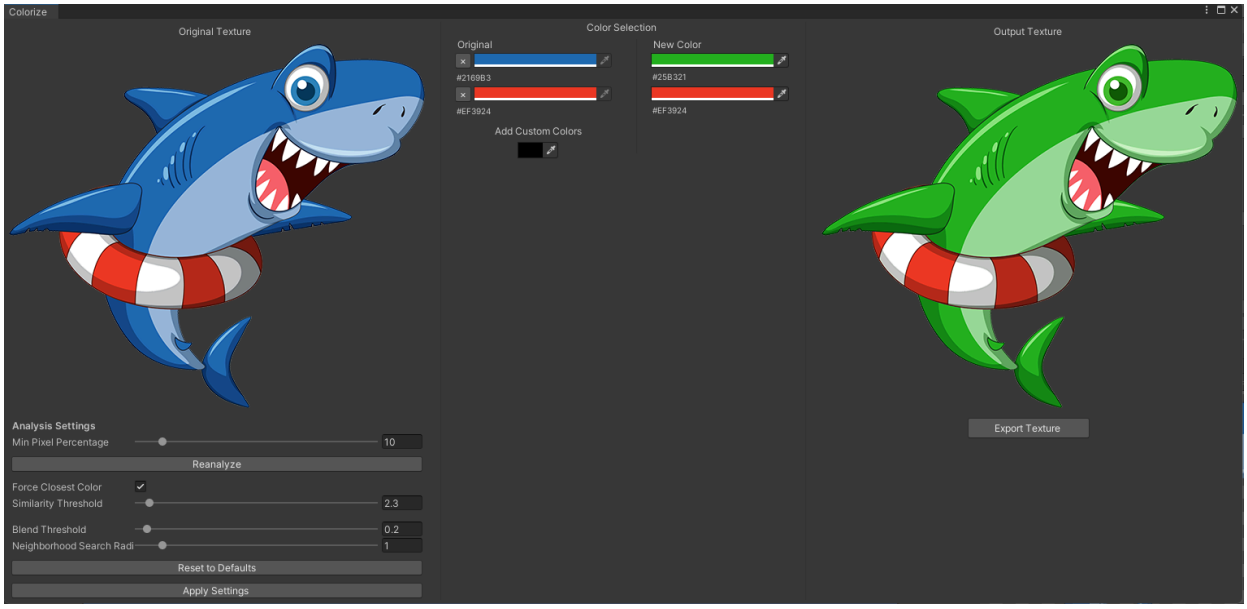
- The right column in the Color Selection section allows you to modify any corresponding colors from the left column. The changes will apply to all pixels containing that same color or similar shades, determined by the **Similarity Threshold** property. Increasing this value broadens the range of colors considered similar.
- A new color is applied to the corresponding pixels by calculating the difference between the original and new color and applying that difference to all affected pixels. This method ensures that recoloring preserves details rather than making all pixels a uniform color.
- When the **Force Closest Color** toggle is disabled, a pixel's color will only be changed if its similarity to the closest color in the left column is below the **Similarity Threshold**. If the pixel's color doesn't meet this criteria, it won't be considered a match for any color in the left column, and its color will remain unchanged when modifying the colors in the right column.
    - For example, the images below show the difference in color changes when using different values for the **Similarity Threshold**. As the threshold increases, it also includes additional shades of the selected blue color.





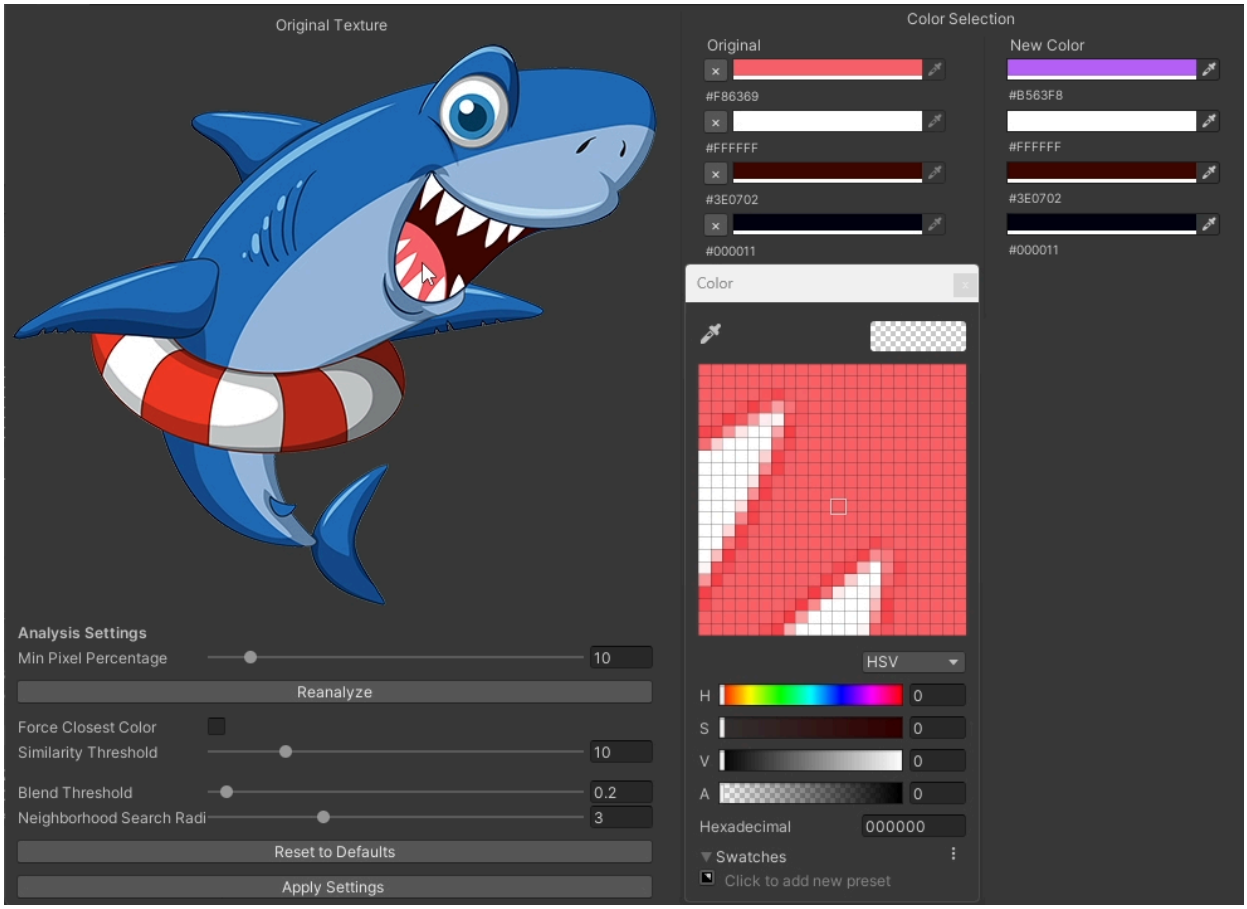
- Enabling the **Force Closest Color** toggle disables the Similarity Threshold, meaning each pixel will be mapped to the closest color from the left column. Misusing this option can lead to unintended results, so refer to Rule of Thumb #3 for proper usage guidelines.
    - For example, the images below show how enabling the Force Closest Color toggle affects recoloring blue shades. In one case, only blue is selected, causing even the red shades to shift toward it. In the other, a red color is also selected, so the red shades are mapped to that color and handled separately from the blue shades.

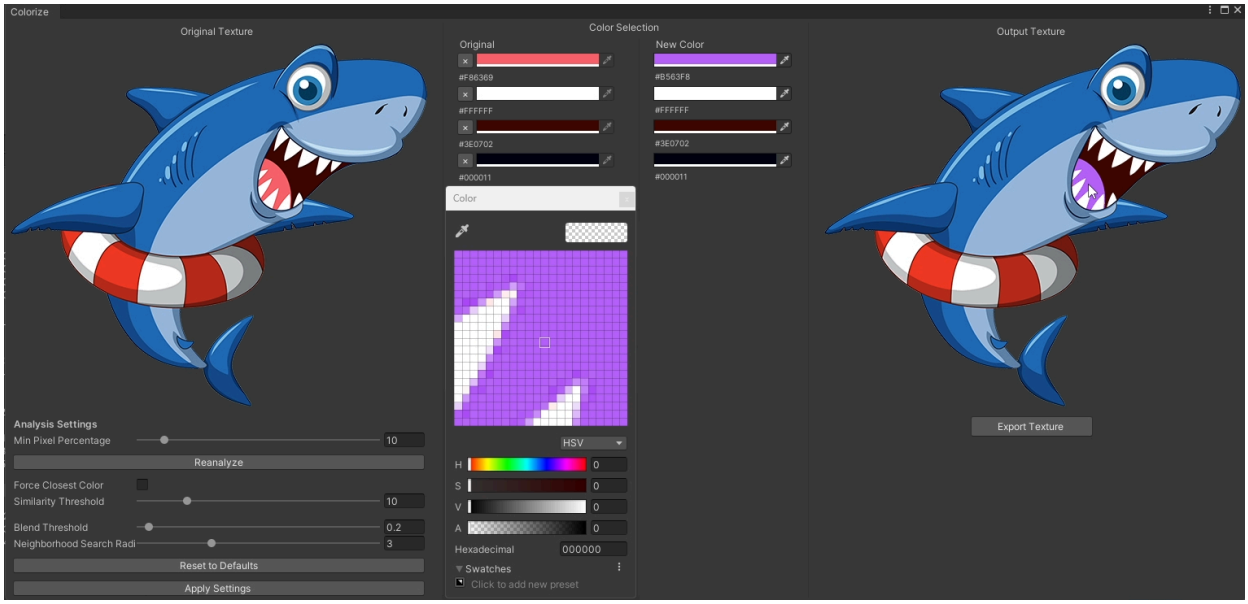




- When two colors meet, blending can occur, creating pixels that are a mix of those colors. When we change any of the dominant colors, we want the blending pixels to be affected as well. This is where the **Neighborhood Search Radius** property comes into play. This property defines how many surrounding pixels will be checked to find a dominant color if a pixel is not similar to any of the colors.
    - Setting this value to 0 means that blending will not be detected at all.
    - Check out Rule of Thumb #4 for more details about using the **Neighborhood Search Radius** property.

- For blending to work correctly, both colors involved need to be included in the left column of the Color Selection section. Even if you don’t want to change one of those colors, include it anyway to ensure that blending and recoloring of the blended pixels work properly.
- The **Blend Threshold** property is used to determine whether a pixel's color is a result of blending between surrounding dominant colors. Increasing this value may cause some unwanted pixels to be considered as blends as well. In most cases, the default value should be sufficient.
- The images below show how blending between red and white can be handled in the sample image.





- When your colorizing is finished and you're ready to use the new sprite, click the **Export Texture** button under the output texture to save it as a **PNG** file and use it in your project.

Rules of Thumb

1. If your texture contains similar colors, use a higher **Min Pixel Percentage** to filter out minor variations. You can then manually add the necessary colors to avoid confusion.
2. If you want to change a minor color, add it manually instead of adjusting **Min Pixel Percentage**. This prevents unnecessary colors from being included and ensures you only modify the ones you intend to.
3. Enabling the **Force Closest Color** toggle maps each pixel to the nearest color from the left column, making it useful when you want to change all shades of a color at once. This works best for images with only a few distinct colors. For example, when recoloring all shades of blue while keeping all shades of red unchanged. However, in images with many different colors, you may need to manually add colors you don’t want to change to the left column to prevent unintended modifications.
4. Increasing the **Neighborhood Search Radius** value expands the number of pixels checked for every pixel that is not similar to the dominant colors. This increase happens quadratically (9 pixels for a radius of 1, 16 for a radius of 2, etc.), so setting a higher value may significantly slow down the recoloring process. For most cases, a value between 0 and 3 should be sufficient.

Contact

- If you encounter any issues or have any questions, please feel free to contact me via email at smahdifaghih2001@gmail.com or connect with me on LinkedIn.

Review

- If you found EasyColorize2D helpful and enjoyed using it, I'd greatly appreciate a quick review and rating on the Asset Store. Your feedback not only helps me improve but also supports other developers in discovering this tool. Thank you for your time and support!