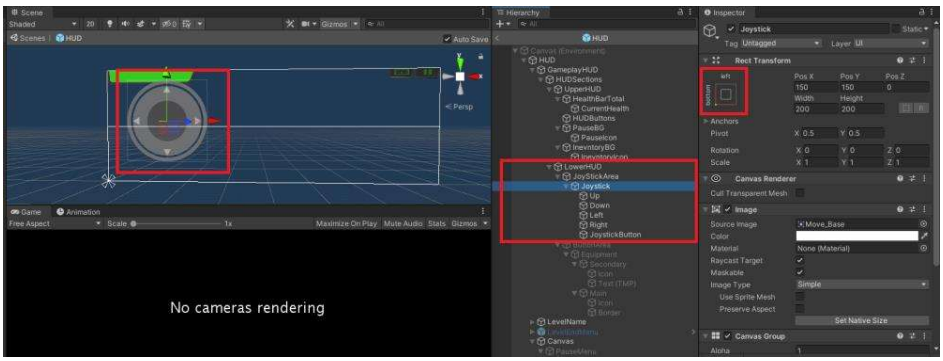
Sprint 02

Assignment 02: Converting to Mobile

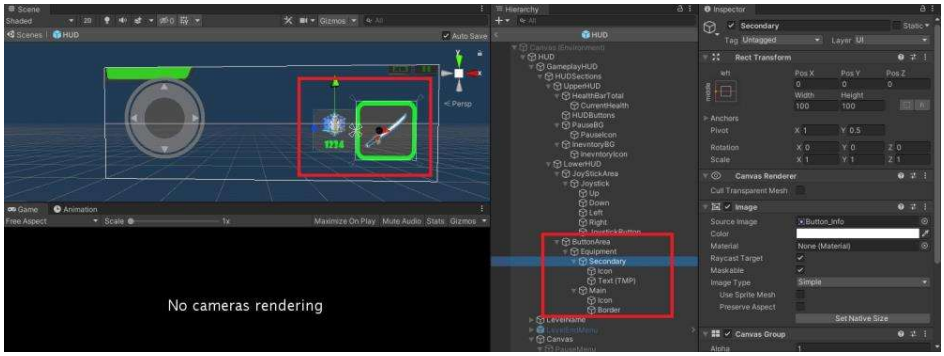
Step 01: Setup

- We first create the UI elements of the Assignments.
- We open the "*HUD*" Prefab and under the "*LowerHUD*" section, we create an Empty GameObject called "*JoystickArea*" and assign to it the "*ValidJoystickArea*" tag.
- And under that, we create an Image called "*Joystick*" and attach the Joystick Image/Sprite to it.
- We also align the Anchor Points if this "*Joystick*" Image, i.e., the Image with Direction Arrows to "*BottomLeft* Corner" point, so as to make it stay to the bottom left side of the screen and scale accordingly.
- Then we add another Image-GameObject called "*JoystickButton*" and add the joystick button image to it i.e., the Black-Sphere.
- We also add to it Unity's Built-in script called "*On-Screen Stick*" and set its "*Control Path*" to "*Left Stick (Gamepad)*".
- Then we add 4 image game objects under the joystick game object and add to it "*The Green Heat-Map*" image and rename them accordingly up, down, left and right.



- Then we create a "*JoystickController*" script and attach it to the joystick button GameObject.
- This script controls the movement of the joystick and the heat maps that are shown by the inputs of the player.
- then we add a reference to the "*JoystickController*" in the "*HUDController*" script.

- In similar way we create the functionality for equipment use button.
- We create an empty GameObject called "*ButtonArea*" under the "*LowerHUD*" GameObject and align its anchor point to top left corner from the anchor presets option.
- Then we create a child GameObject called "*Equipment*" and attach "*CanvasGroup*" component to it.
- Then under it we create 2 Image-type GameObjects, called "*Main*" and "*Secondary*".
- To the "*Main*" GameObject, we add Unit's Built-in Script called "*On-Screen Button*" and set its "*Control Path*" to the Keyboard Key "*E*".
- We do not add this script to the "*Secondary*" GameObject because we simply do not use it and it is kept just for the purpose of visual representation.
- Under the "*Main*" GameObjects, we create two more GameObjects called "*Icon*" and "*Border*" and asign the respective images of icon and border.
- Similar way under the secondary game object we create two child GameObjects called "*Icon*" and "*Text*" and assign the respective values to them.

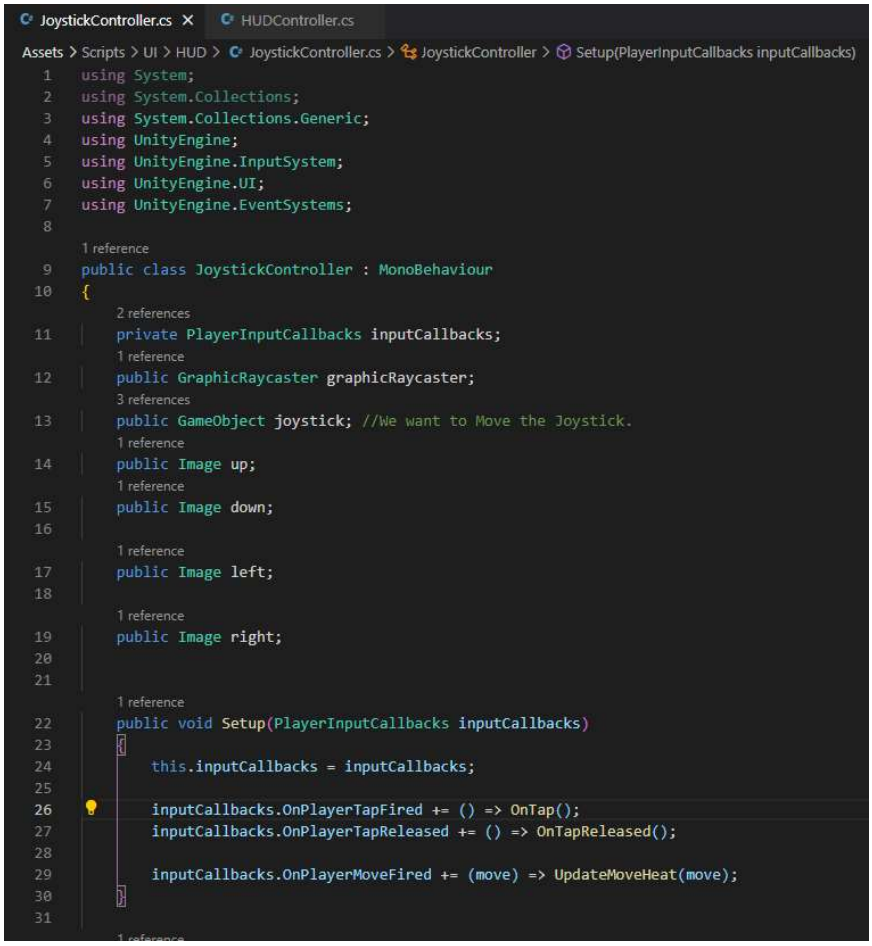


Step 02: Script & Workflow

- The Scripts workflow is like this:
    - HUDController > JoystickController.

❖ **JoystickController.**

- This script performs the action off taking the player inputs and moving the player with respect to the changing position of the joystick button.
- Also, this script executes the functionality to move the entire "*Joystick*" GameObject to wherever the Player clicks on the "ValidJoystickArea" on the Screen.
- This script also has a function that that Updates the Heat-Map on the Joystick, with respect to the inputs by the player.
- in addition to that it also has functionality be that the joystick GameObject changes it's alpha value when the player clicks on the screen and when the player does not click on the screen.

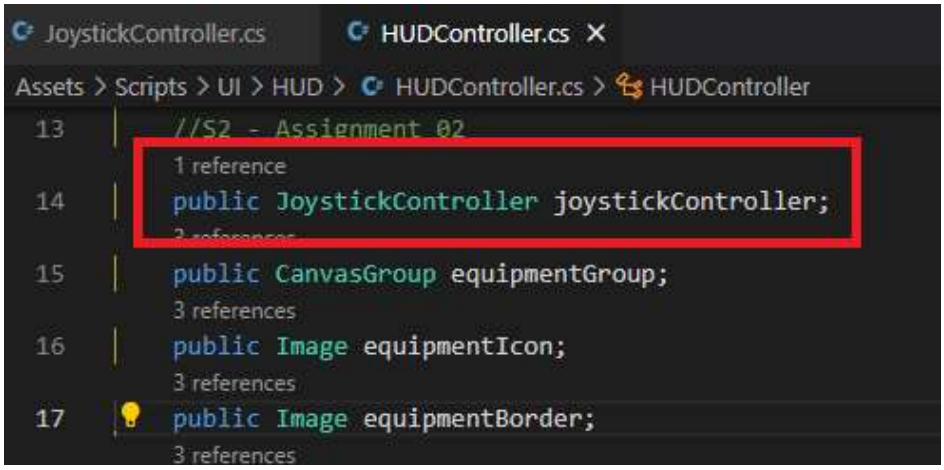

JoystickController.cs
HUDController.cs
Assets > Scripts > UI > HUD > JoystickController.cs > JoystickController > Setup(PlayerInputCallbacks inputCallbacks)
using System;
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.InputSystem;
using UnityEngine.UI;
using UnityEngine.EventSystems;
1 reference
public class JoystickController : MonoBehaviour
{
2 references
private PlayerInputCallbacks inputCallbacks;
1 reference
public GraphicRaycaster graphicRaycaster;
3 references
public GameObject joystick; //We want to Move the Joystick.
1 reference
public Image up;
1 reference
public Image down;
1 reference
public Image left;
1 reference
public Image right;
1 reference
public void Setup(PlayerInputCallbacks inputCallbacks)
{
this.inputCallbacks = inputCallbacks;
inputCallbacks.OnPlayerTapFired += () => OnTap();
inputCallbacks.OnPlayerTapReleased += () => OnTapReleased();
inputCallbacks.OnPlayerMoveFired += (move) => UpdateMoveHeat(move);
}

```
1 reference
private void UpdateMoveHeat(Vector2 movement)
{
    Color full = Color.white;   //Full Alpha

    //UP
    full.a = movement.y;    //a: Alpha component of the color.
    up.color = full;

    //DOWN
    full.a = -movement.y;
    down.color = full;

    //RIGHT
    full.a = movement.x;
    right.color = full;

    //LEFT
    full.a = -movement.x;
    left.color = full;

}
1 reference
private void OnTap()
{
    //Mouse Pointer Setup
    Vector2 mousePosition = Pointer.current.position.ReadValue();
    PointerEventData ped = new PointerEventData(null);
    ped.position = mousePosition;   //Assign MousePosition to the PED.

    List<RaycastResult> results = new List<RaycastResult>();    //This holds a List of RayCasts that occured, when we Touch the Screen - Space.

    //The Graphic Raycaster is used to raycast against a Canvas.
    //The Raycaster looks at all Graphics on the canvas and determines if any of them have been hit.
    graphicRaycaster.Raycast(ped, results);
    Debug.Log("OnTap");

    foreach(RaycastResult result in results)
    {
        if(result.gameObject.tag.Equals("ValidJoystickArea"))
        {
            Debug.Log("GraphcisRaycaster");

            ActivateJoystick(result.screenPosition);    //Position on the Screen where we Tap.
        }
    }
}
```

```
1 reference
private void ActivateJoystick(Vector2 hitScreenPosition)    //Move the Joystick anywhere along JoyStickArea.
{
    joystick.transform.position = hitScreenPosition;
    joystick.GetComponent<CanvasGroup>().alpha = 1.0f;
}

1 reference
private void OnTapReleased()
{
    inputCallbacks.OnPlayerMoveFired(Vector2.zero);   //Safety - MoveFired holds the Player Movement Control.
    joystick.GetComponent<CanvasGroup>().alpha = 0.3f;
}
}
```

❖ **HUDController.**

- In this script, we added a reference to the function "*Setup*" up from the "*JoystickController*" and execute it respectively.



- Then in the same script we add functionality to use the "*Main*" and "*Secondary*" Equipped Weapon.
- First, we create a function called "*UpdateEquipmentEnabledState*" and call it in the "*Update*" function.
- Then we add an If-Else statement and check if the Player can interact with the object, and if it is true then we set the "*Equipment*" GameObject to active.
- If not, then we add some other check functionalities in the Else statement.
- Then we add an If statement where we check if they Equipped Weapon can be used as Primary, and if it returns true, then we set its Icon and Border to Active and if not, then we set its Icon and Border to Desaturated and Inactive respectively.

- Then we exit from the Else statement and add another If-Else statement where we check if the Secondary Equipment is Null then we set its Alpha to 0 (Zero) and if it is not Null then we set it's alpha to 1.
- Then we exit from the function and add another function called "*UpdateSecondaryEquipment*".
- Then in the "*UpdateSecondaryEquipment*" function we add an if statement where we check if the Secondary Equipped Item is not Null then we set the icon of that respective Item Type to Active and Set the Amount or Number of that Item present in the Equipment, in the Text Component Area, which we get from the "*GetCount*" function, from "*InventoryController*".
- Then after setting up this function, we subscribe to this function from "*SetupHUDController*" function.
- We Subscribe to this function with the Action Events: ***OnNewItemEquiped***, ***OnItemConsumed***, ***OnPickupEventCollected*** and also inside the "***SetupHUDController***" function, just in case it doesn't go or subscribe to any of the above functions.

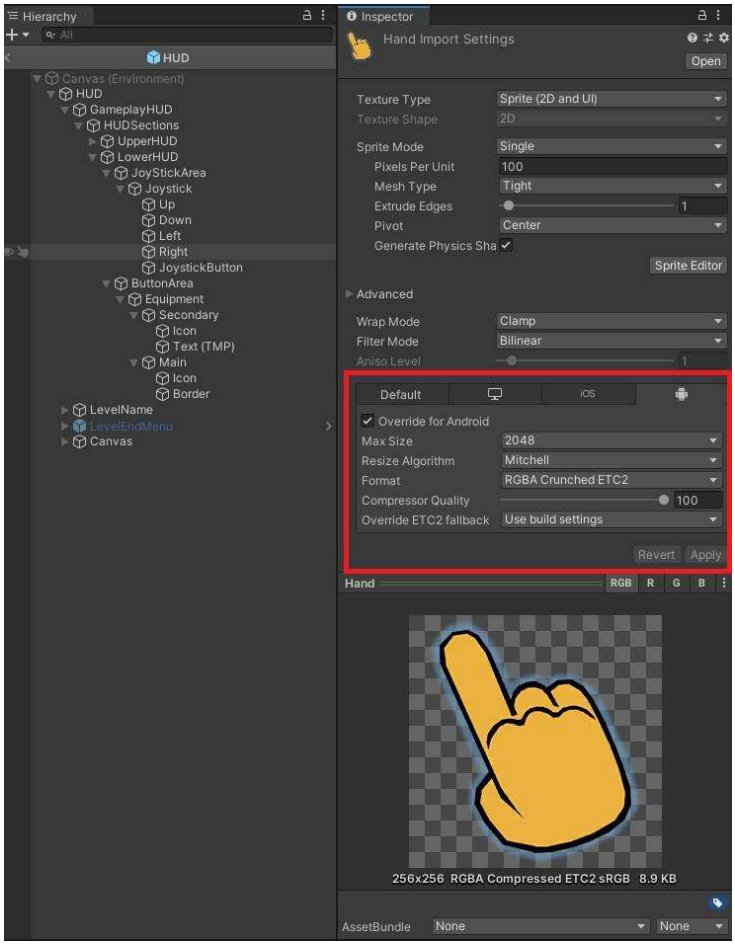
```
0 references
public void Update()
{
    UpdateEquipmentEnabledState();
}

1 reference
private void UpdateEquipmentEnabledState()
{
    if(player.Interaction.CanInteract())    //Hand Icon
    {
        equipmentIcon.sprite = interactIcon;
        equipmentBorder.enabled = true;
        equipmentGroup.alpha = 1;
        secondaryGroup.alpha = 0;
    }
    else
    {
        if(player.playerEquipment.PrimaryEqiupment == null)
        {
            equipmentGroup.alpha = 0;
        }
        else    //Primary or Main Icon - Active & In-Active.
        {
            if(player.playerEquipment.PrimaryEqiupment.CanUse())
            {
                equipmentIcon.sprite = icons.GetIcons(player.playerEquipment.PrimaryEqiupment.Type);
                equipmentBorder.enabled = true;
            }
            else
            {
                equipmentIcon.sprite = icons.GetDesaturatedIcons(player.playerEquipment.PrimaryEqiupment.Type);
                equipmentBorder.enabled = false;
            }

            equipmentGroup.alpha = 1;
        }
    }

    if(player.playerEquipment.SecondaryEquipment == null)
    {
        secondaryGroup.alpha = 0;
    }
    else
    {
        secondaryGroup.alpha = 1;
    }
}
```

```
4 references
private void UpdateSecondaryEquipment()
{
    if(player.playerEquipment.SecondaryEquipment != null)
    {
        secondaryIcons.sprite = icons.GetIcons(player.playerEquipment.SecondaryEquipment.Type);
        secondaryText.text = inventoryController.GetCount(player.playerEquipment.SecondaryEquipment.Type).ToString();
    }
}
}
```

```
1 reference
public void SetupHUDController(Player player, InventoryController inventoryController, PickupEvents pickupEvents)  //S2 - Assignment 02
{
    this.player = player;
    this.inventoryController = inventoryController;     //S2 - Assignment 02

    //S2 - Assignment 02
    joystickController.Setup(player.Broadcaster.Callbacks);

    player.playerEquipment.OnNewItemEquiped += UpdateSecondaryEquipment;
    player.playerEquipment.OnItemConsumed += (ItemType) => UpdateSecondaryEquipment();  //Bcoz it is an "Action<ItemType>" delegate which takes a Parameter.

    pickupEvents.OnPickupEventCollected += (pickupEventsp) =>   //S2 - Assignment 02
    {
        UpdateSecondaryEquipment();
    };

    UpdateSecondaryEquipment(); //COnstant Update - id the above 2 conditions doesn't meet, then this Func. will execute.
}

2 references
public void ShowHUD()
{
    gameplayHUDCanvasGroup.alpha = 1f;
}
```

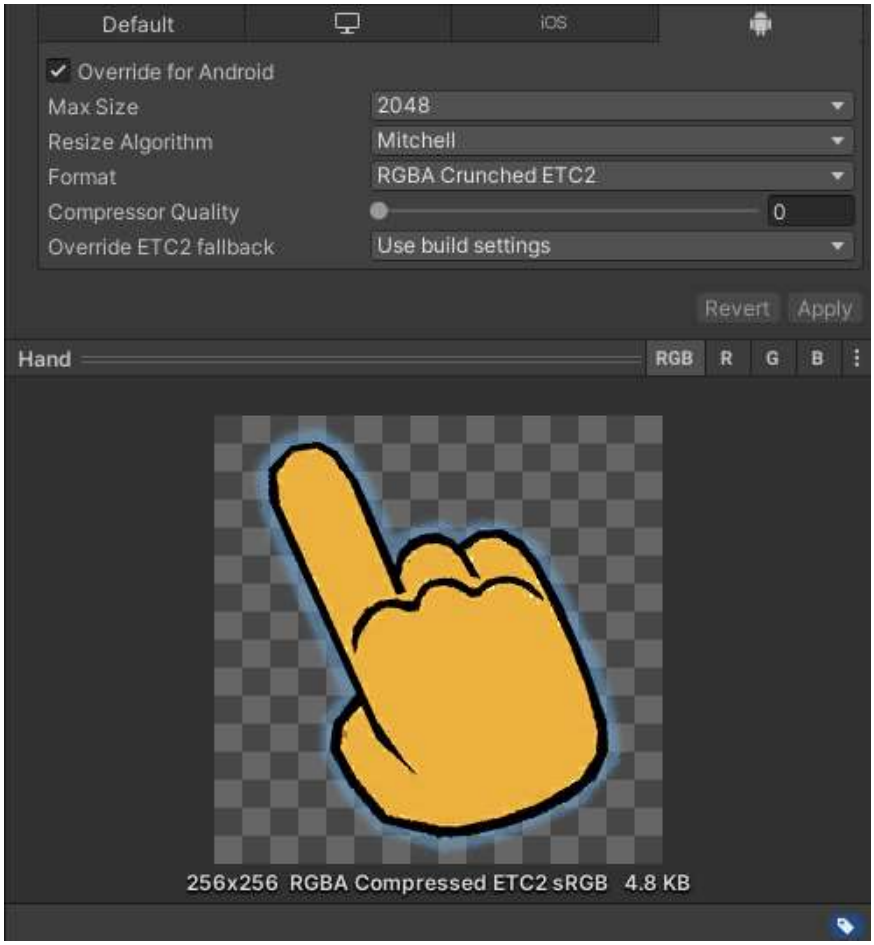
❖ **Setting up Light-Maps & appropriate Image Compression.**

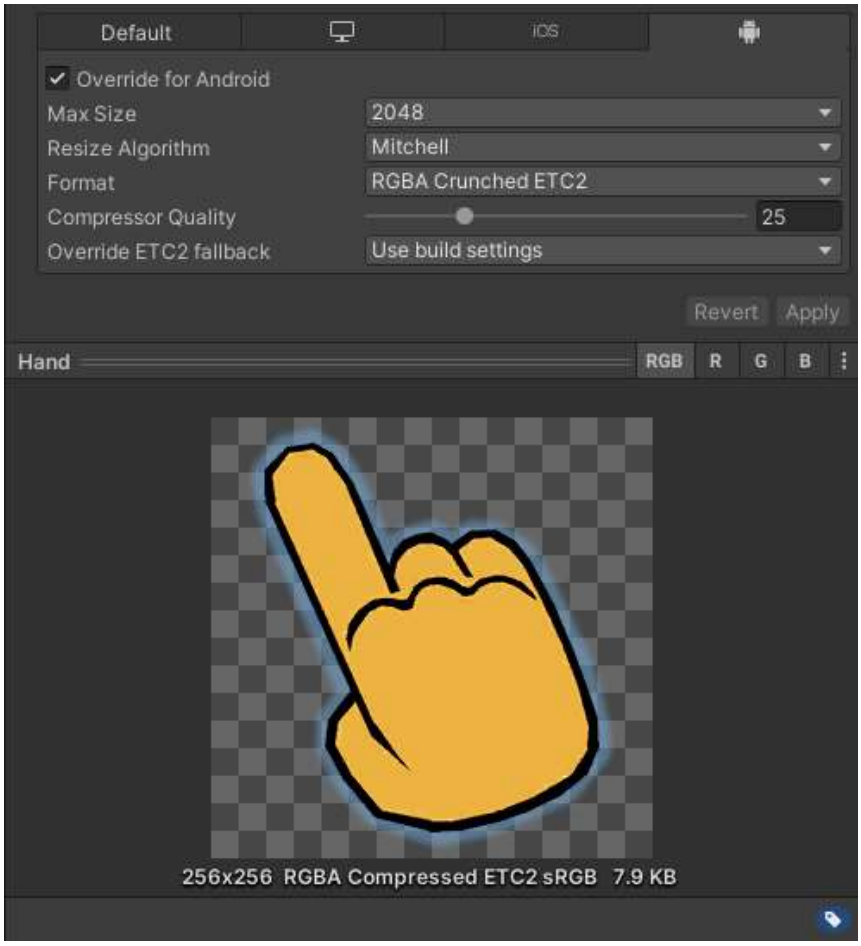
- First, we set the Image Compression format of all images to "*RGBA Crunched ETC2*".
- To do this we simply choose any image, in this case we use the "*Hand*" image or Sprite and set its "Format" to "*RGBA Crunched ETC2*".

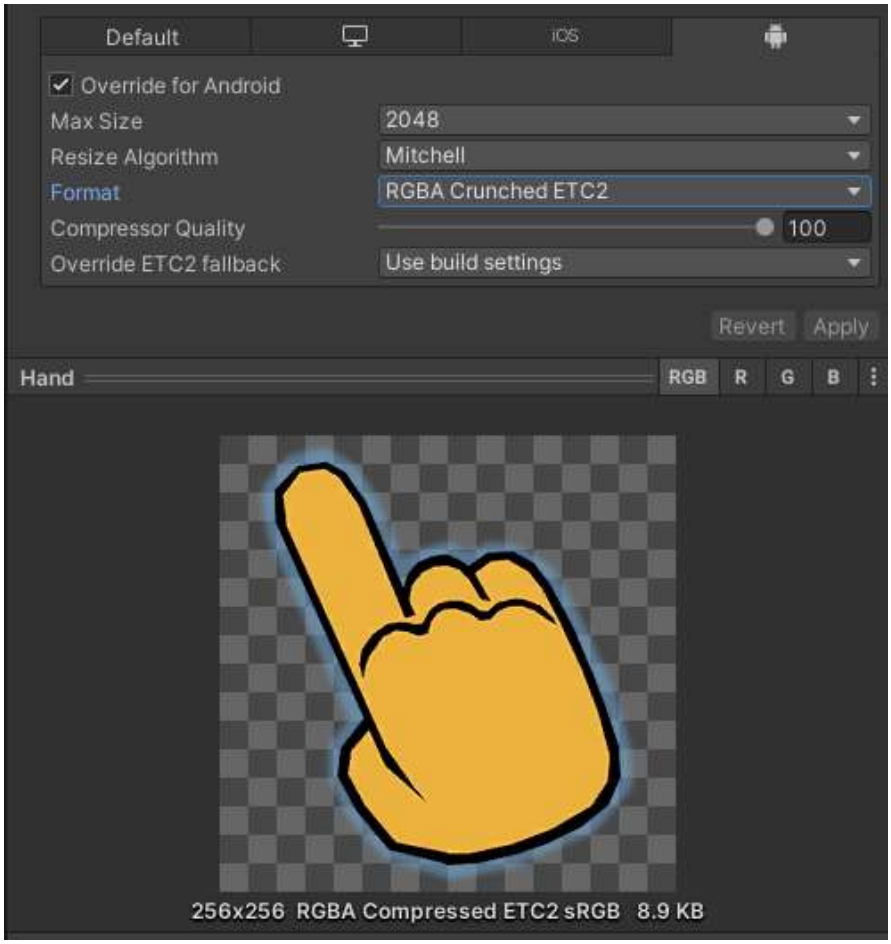


- We set it to this because most of our images are Alpha based, i.e., Transparent, and it reduces the Size of the Image to Save Memory to enhance the Performance.

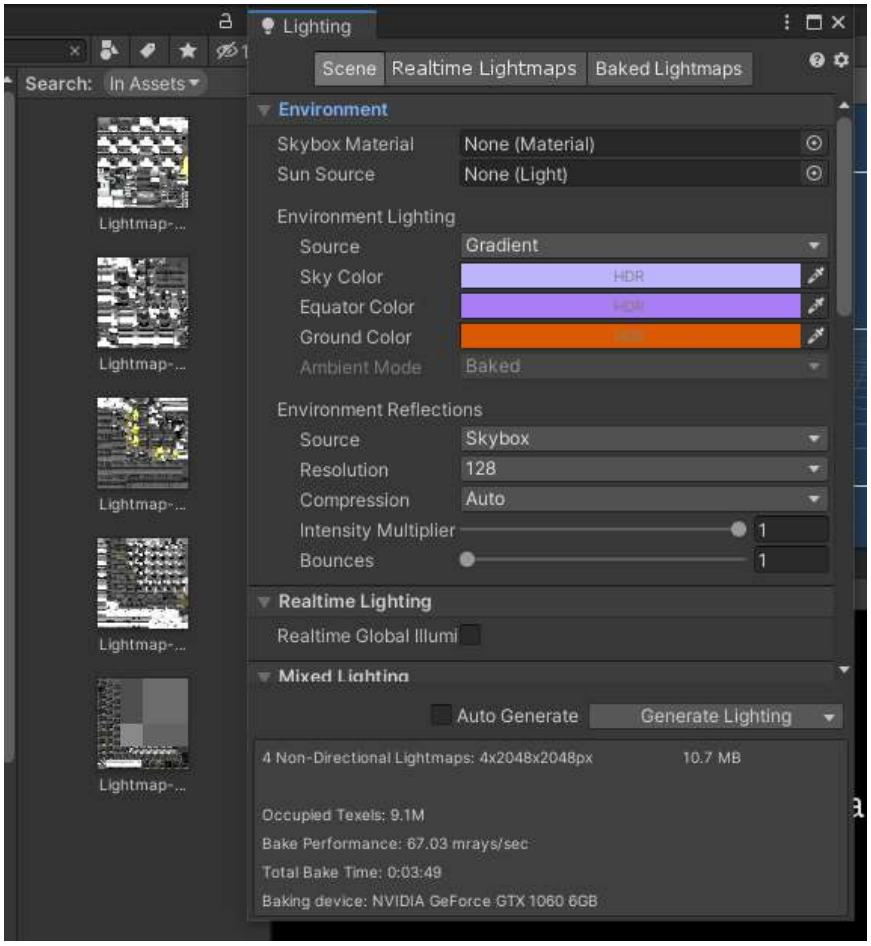
- When we set "*Compressor Quality*" to certain amounts, we get various Image Sizes respectively.
- The less we set the amount of Compressor, i.e., close to 0, we get lesser Image Size and Low Image Quality.
- The Higher we set the amount of Compressor, i.e., close to 100, we get Higher Image Size and better Image Quality.




Default
iOS
Override for Android
Max Size
2048
Resize Algorithm
Mitchell
Format
RGBA Crunched ETC2
Compressor Quality
25
Override ETC2 fallback
Use build settings
Revert
Apply
Hand
RGB
R
G
B
256x256 RGBA Compressed ETC2 sRGB 7.9 KB



- There are many other Image Compression formats.
- One related to the ETC2 is the "*RGB Crunched ETC*" which ignore the Alpha and compresses the Image with a Solid Background.
- Then after that, we set the Light-Maps.
- We open the "*Lighting Settings*" from Windows > Rendering > Lighting Settings.
- Then we simply click on the "*Generate Lighting*" button and generate the Light-Maps.



◊ **Extras:** *Crunch* is a lossy texture compression format, which is normally used on top of DXT texture compression. Crunch compression helps to reduce the size of the textures in order to use less disk space and to speed up downloads. **Ericsson Texture Compression** (**ETC**) is a lossy texture compression technique developed in collaboration with Ericsson Research in early 2005.

Step 03: What have I learnt

- **Crunch compression reduces the size of a texture on disk, what are some of the drawbacks of using it?**

– It takes Time to Compress.
– It is File Dependent.
– Files need to be Compressed before loaded into Memory.
– It can reduce the Quality of the Image after Compression.
– The Size of Image after Compression depends on the Crunch Quality.
– Higher the Crunch Quality, higher the Files Size & Quality of the Image.
– Lower the Crunch Quality, Lower the Files Size & Quality of the Image.

- **Briefly explain the difference between ETC and ETC2 in compatibility and quality.**

– ETC doesn't support Alpha Channel while ETC2 supports it.
– ETC doesn't support POT (i.e., it is NPOT) while ETC2 supports POT.
– ETC only supports OpenGL 3.0 and less while ETC2 supports OpenGL 3.0 and above with Backward Compatibility.
– ETC2 gives better Image Quality for the Same File Size than ETC.

◊ **Extras:** A *POT (Power of Two)* texture is where a texture has a resolution of 2^n x 2^n. It is a *Seamless Texture*. And *NPOT (Not Power of Two)* is not a *Seamless Texture*.

----------THE END----------