Table of Contents

Cooking Toolkit

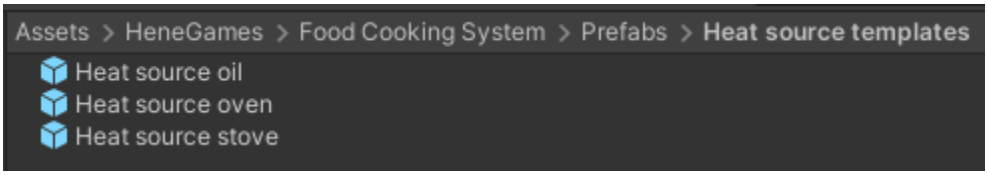
Online **Documentation**

Remember to see the example scenes from
Assets/HeneGames/FoodCookingSystem/ExampleAssets/Scenes

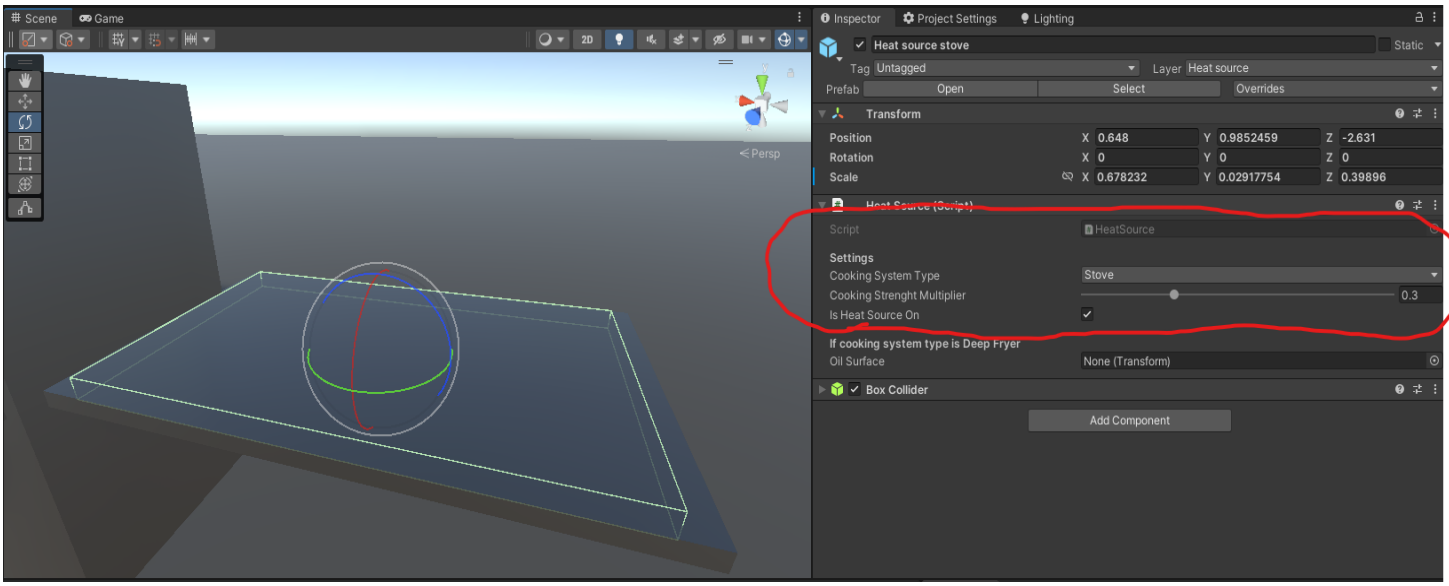
With this asset you can cook the colors of the vertices. The shader that comes with the asset shows a visual appearance.

- **Red vertex color** is the mask of frying.
- **Green vertex color** is a mask for deep frying.
- **Blue vertex color** is a buffer from frying to burning.
- **Alpha vertex color** is a mask of burning.

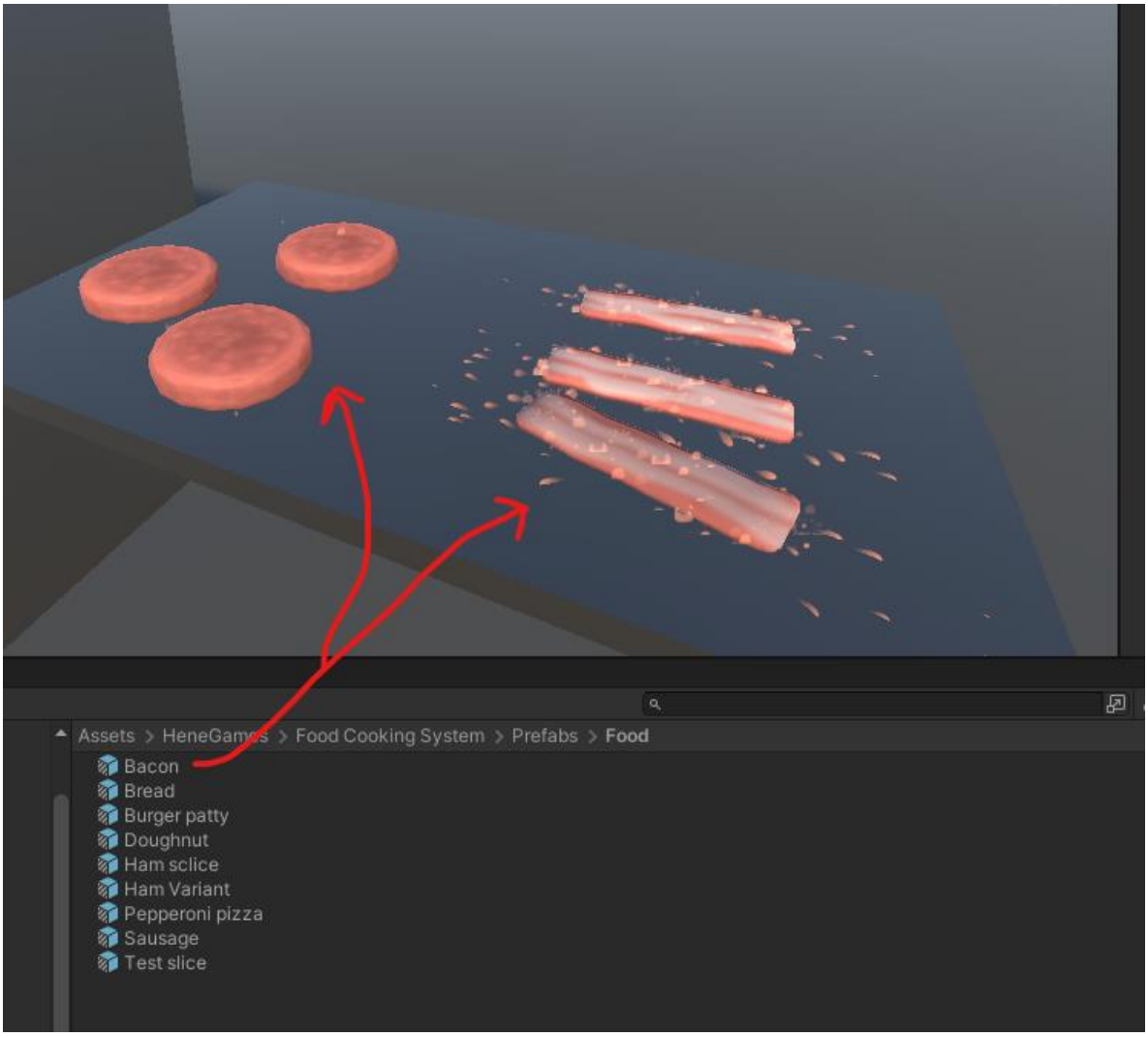
Overview



The working principle is easy, you must drag a **heat source prefab** to your scene. You can **scale** the prefab to the size you want.



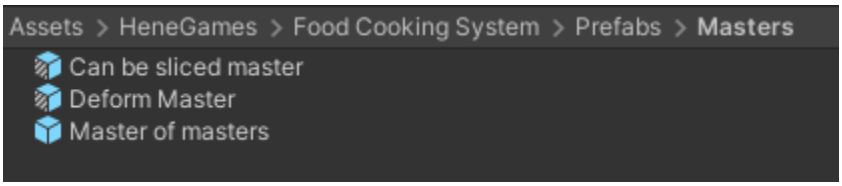
You can adjust the intensity of the **heat source** and whether it is on from these.



Drag ready-made prefab to your scene. All food prefabs inside the box collider start cooking if the **heat source** is on.

Select the **food prefab** and change the values in the inspector tab as you wish.

Master Prefabs



With the help of these prefabs, new food assets are made. Never delete, change or copy these prefabs; they are intended for inheritance only (So I mean **prefab variants**).

Making New Food Assets

YouTube Tutorials

- **Make food prefab**
- **Make sliceable food prefab**
- **Make shape-changing food prefab**
- **Make food prefab from someone asset**

Editor Values

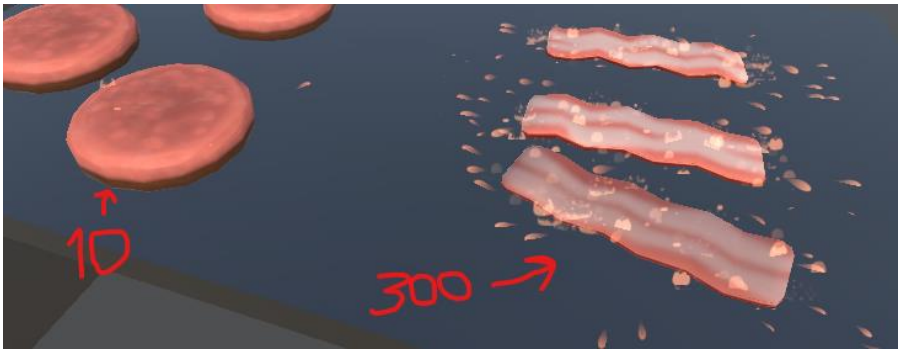
Deep Frying Settings

Floating Force: Determines how much force is given upwards when the food prefab floats inside the **Heat source oil** prefab. The rigid body weight of the food affects this value.

Float When Deep Frying: Determines whether food floats when fried inside the **Heat source oil** prefab.

Floating Depth: Generic value with which you can adjust the floating depth when the prefab is inside the **Heat source oil** prefab.

Cooking Settings



Fat Content: This value determines how many particles the **Cooking Effect** emits, when the food is cooked.

Cooked Scale Multiplier: This value determines the size change of the food when cooking. A value of 0.2 signifies +20% of the original size, while -0.2 indicates a decrease of 20%. You can also set this value to zero to maintain the original size.

Cooking Speed Settings

Cooking Speed: Adjusts the cooking speed of the food. This variable isn't tied to any specific unit; experimentation will determine the optimal value.

Cooked To Burned Speed: Once the vertex is fully cooked, a timer initiates. This value controls the speed at which the timer counts down before the vertex starts to burn. It acts as a buffer to prevent immediate burning after cooking.

Cooking Percent Settings

Cooked Enough Percent: 0 = 0%, 1 = 100%. When the **Cooked Percent** value surpasses or matches this threshold, the food is considered ready.

Burned Over This Percent: 0 = 0%, 1 = 100%. When the Cooked Percent value surpasses or matches this threshold, the food is considered burned.

Stove Cook Top Side Percent: This variable enables you to determine the maximum cooking percentage of vertices above the food mesh center point that become fried while cooking on the stove. For instance, setting this variable to 0.25 implies that all vertices from the center and above will be fried by 25%, even those not facing the stove directly. This adjustment compensates for the fact that heat partially travels upward through the food when it's fried on the stove.

Food Name

You can put the name of the food here. If this is left blank, the name will be generated automatically in the **Awake** function.

Debug Variables

Never change these variables in the **editor**. These values exist only so that you can easily see what is happening to the food in the editor.

Use In Code

Of course, this asset would be nothing if you couldn't easily ask the **Food.cs** script what happens inside it. You need a reference somehow for the **Food** class and you can ask it for public variables.

```
using HeneGames.CookingSystem;
using UnityEngine;

Unity Script | 0 references
public class Example : MonoBehaviour
{
    //Foof class reference
    [SerializeField] private Food food;

    Unity Message | 0 references
    private void Update()
    {
        //Boolean
        food.IsOnHeatSource();

        //Boolean
        food.ItsCooking();

        //Boolean
        food.IsReady();

        //Boolean
        food.IsBurned();

        //Float from 0f - 100f with one decimal accuraty
        food.CookedPercent();

        //Float from 0f - 100f with one decimal accuraty
        food.BurnedPercent();

        //Vector3 position
        food.CenterOfMass();

        //Reference to the heat source where the food is
        food.CurrentheatSource();
    }
```

Here are the variables you can easily ask from the **Food** class.

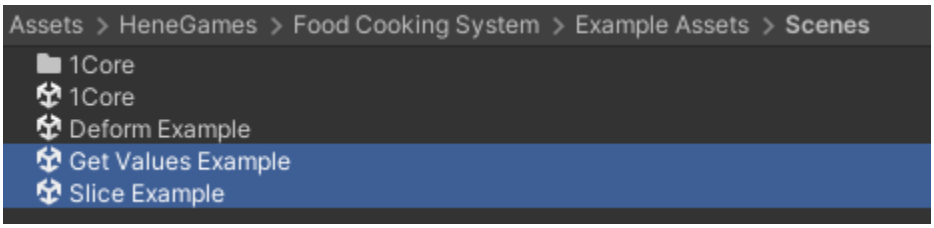
```
using HeneGames.CookingSystem;
using UnityEngine;

Unity Script | 0 references
public class Example : MonoBehaviour
{
    //Heat source class reference
    [SerializeField] private HeatSource heatSource;

    Unity Message | 0 references
    private void Update()
    {
        //List of all foods that are inside this heat source
        heatSource.FoodList();
    }
}
```

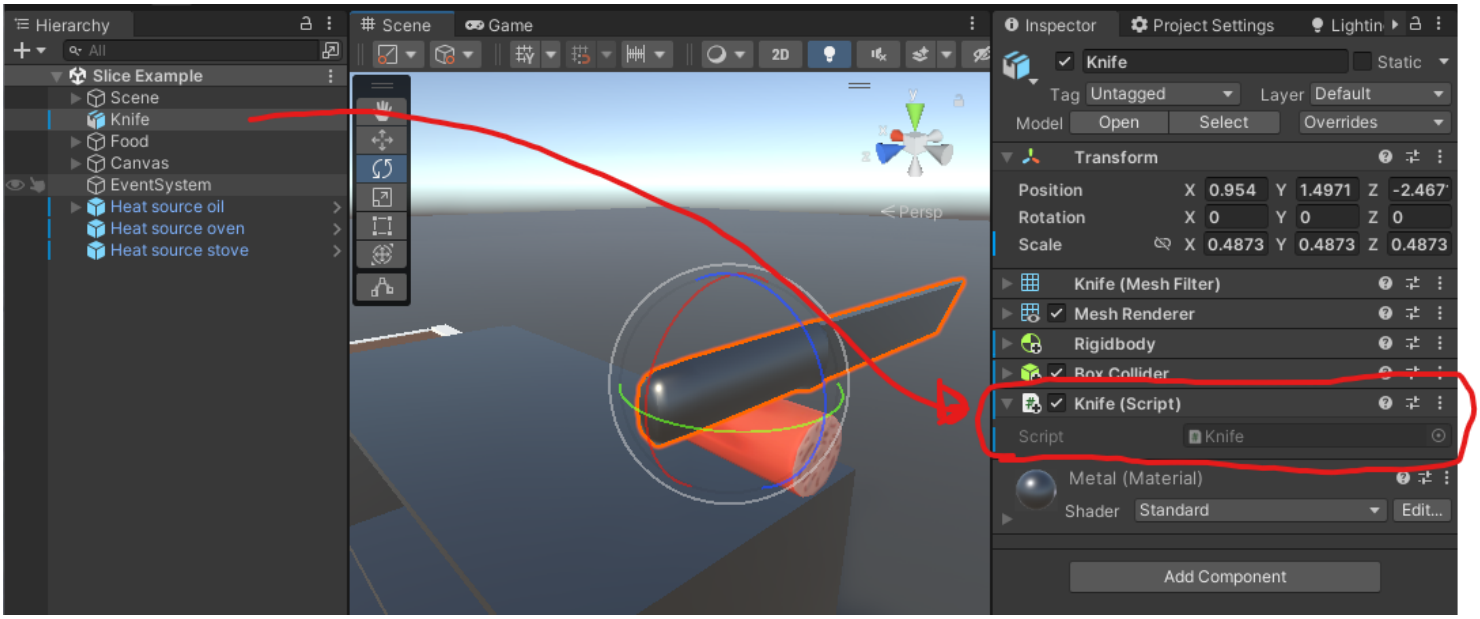
You can also ask the **HeatSource** class for a list of all the foods in it.

Example Scenes



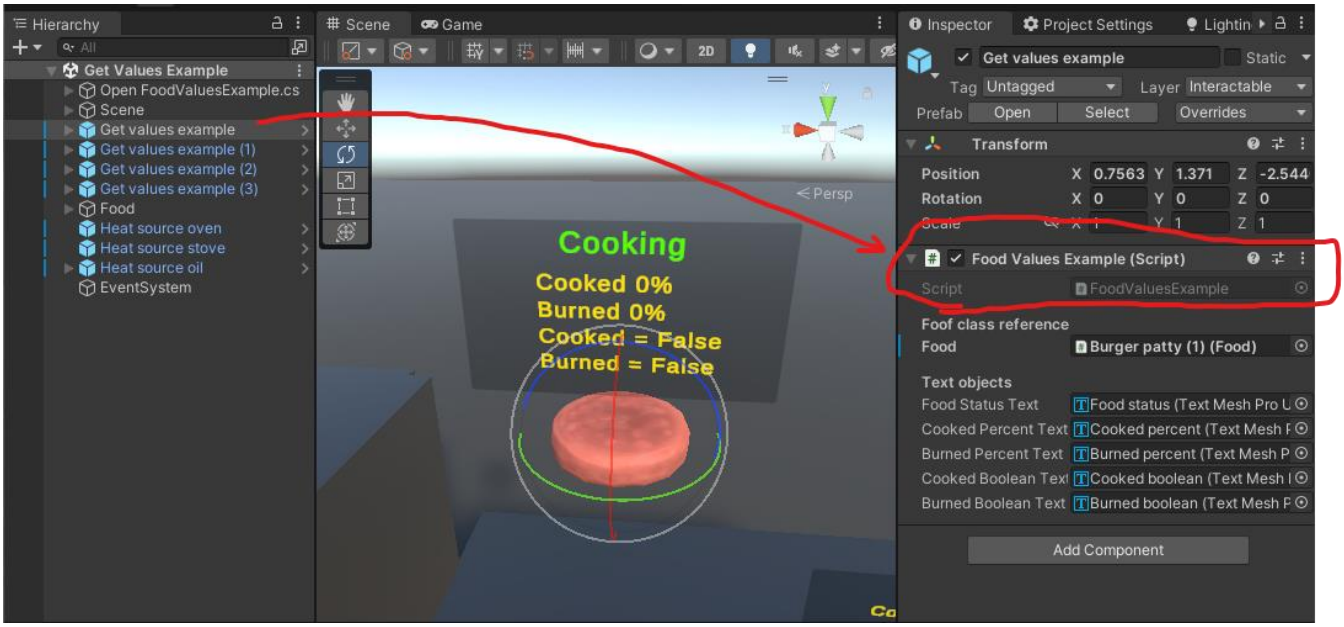
If you want some kind of coding example, I recommend looking at these example scenes.

Slice Example



In the **Slice example**, you should open the **Knife script** and see how it's done.

Get Values Example



In the **Get Values Example**, you should open the **FoodValuesExample script** and see how it's done.