# Loot Labels 1.1

## Introduction:

Hi everyone.
In order to raise some money to fund my own game, Setting Sun, I decided to start converting some features of it to separate modules to sell them on the asset store.

Since the game is loot based I wanted to have labels above the loot that stack on each other instead of overlapping.
This makes it easier to see what dropped, to choose what you loot, etc...

## What it is:

This asset **creates a label for each in game object** you want.
You can either give your object a normal label, or a stacklabel which causes it to stack on other stacklabels.

## Roadmap link:

Trello

## Forum link:

Forum

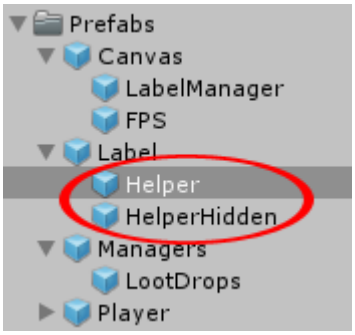
## Changelog 1.1:

- Normal labels and stack labels merged into 1 object
- Added an option to switch text with icons
- Added a range checker which queues the interaction and makes your player run to the object
- Added an option to loot by walking over the object
- Updated player navigation
- Lots of code refactoring and bugfixes
- Expanded documentation
- Packages for new unity versions

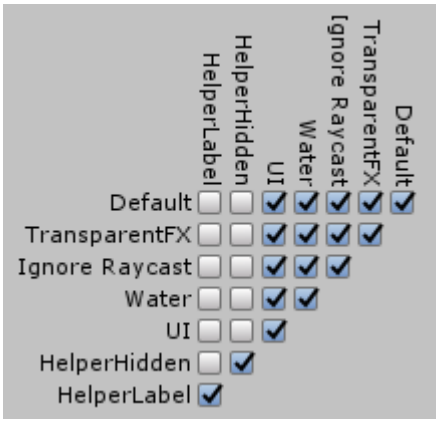
# Getting Started

## Initial setup:

1) Drag the Canvas from the prefabs folder in the hierarchy
2) Add an UI/EventSystem to the hierarchy if you don't have it yet
3) Drag the Managers object from the prefabs folder on the canvas
4) Add a physics raycaster to the main camera
5) Add the following 2 layers to the layer list: HelperLabel, HelperHidden (**needs to be correct, with capitals, no spaces!)**
6) In the prefabs folder assign the HelperLabel layer to Label/Helper object (Prompt: this object only) and the HelperHidden layer to the Label/HelperHidden object




7) Go to Edit/Project Settings/Physics 2D and set the Layer Collision Matrix as shown in this picture for the layers you just added.



## Preparing your object

1) Drag the DroppedCurrency, DroppedGear, Container or NPC script (Scripts/Objects/Loot and NonLoot folder) to your objects parent transform. (See additional notes)
2) Add a preferred collider to your object or one of its children to catch the mouse events and check IsTrigger.
3) Assign which part of your object you want to follow in the “object to follow” slot of the Create Label script. (itself or a specific child)
4) In the ObjectHighlight script assign your outline shader
    a. If your object has a renderer on the parent check Highlight This
    b. If you object has no renderer on the parent:
        i. Drag the MeshHighlight script (Script/Objects/Highlighting folder) on **each child object with a renderer that you want highlighted** when mousing over.
        ii. Drag the VisibilityEventsChild script (Script/Objects folder) to 1 of the children with a renderer. **We need this script to know when the object is no longer visible to the camera!**
        iii. Assign the event handler script from the parent to the slot of the VisibilityEventsChild script
5) Check the settings, fill in the text and assign the colors you want the label to have and voila! (Settings info at the bottom)

## Additional notes

I made a couple of example scripts for the following types of objects: NPC, container, DroppedGear and DroppedCurrency. These are basically your item’s scripts that contain the functionality to do what you want them to do.

The **droppedGear script** contains a reference to the baseGear class (which would contain your item’s stats f.e.) and creates a label based on the rarity given to the baseGear class.
The **droppedCurrency script** does the same as droppedgear, but with a manual chosen color.
The **NPC script** creates a label based on the name and color you provide manually.
The **container script** is the same as the NPC script with additional logic for generating loot.

For more specific objects you can inherit each object and override the mouse functions or just use a different script all together, as long as you call this code and provide the mouse functions, text and the icon path between the brackets.

```
//Called in animator or in start when there is no animator
public override void SpawnLabel() {
    GetComponent<EventHandler>().ClearDelegates();
    GetComponent<EventHandler>().SubscribeMouseEvents(MouseDownFunction, MouseEnterFunction, MouseExitFunction);

    if (currency != null) {
        GetComponent<CreateLabel>().SpawnLabelByColor(currency.ItemName, currency.IconName);
    }
    else {
        Debug.Log("Turn on testing in the dropped loot script or assign currency on instantiate");
    }


    StartCoroutine(GetComponent<EventHandler>().VisibilityCoroutine());
}
```

## Setting up the range checker

This only works with nav mesh/nav agents atm.

1) Assign the player tag to your player
2) Assign the setPlayer script to your player (Automatically adds a navmesh Agent)
3) If you want to loot by walking over a object, add a collider and a kinematic rigidbody to your player
4) Assign CheckForTerrain script to terrain
5) Make sure your navmesh is built
6) Enable range checker in the nav manager and set your minimum interact range

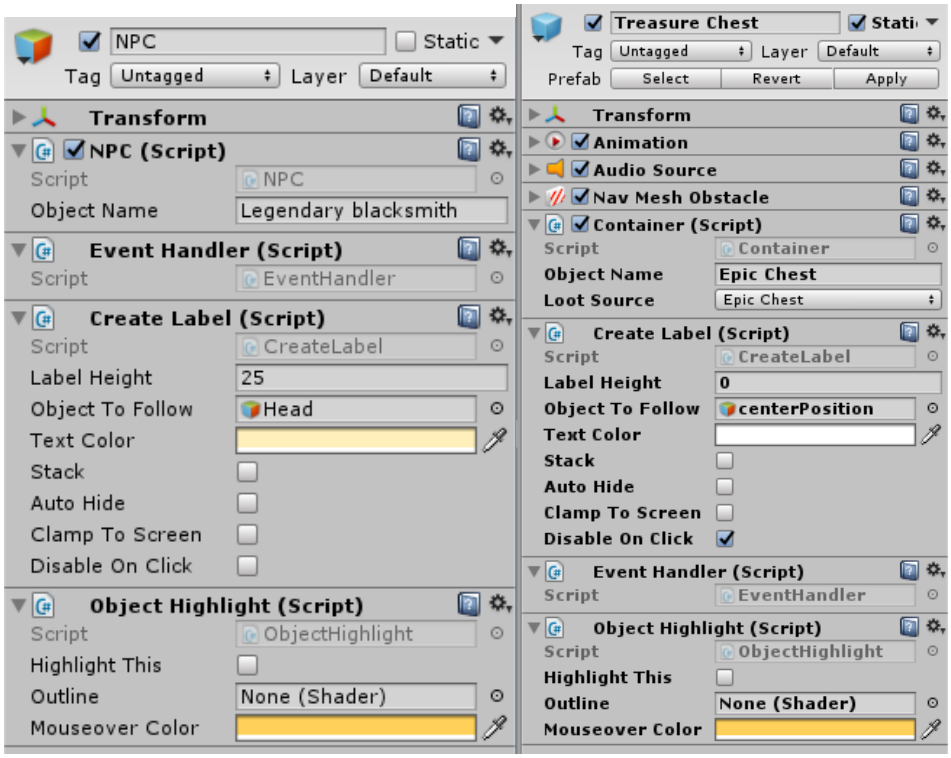
## Expanding loot drops

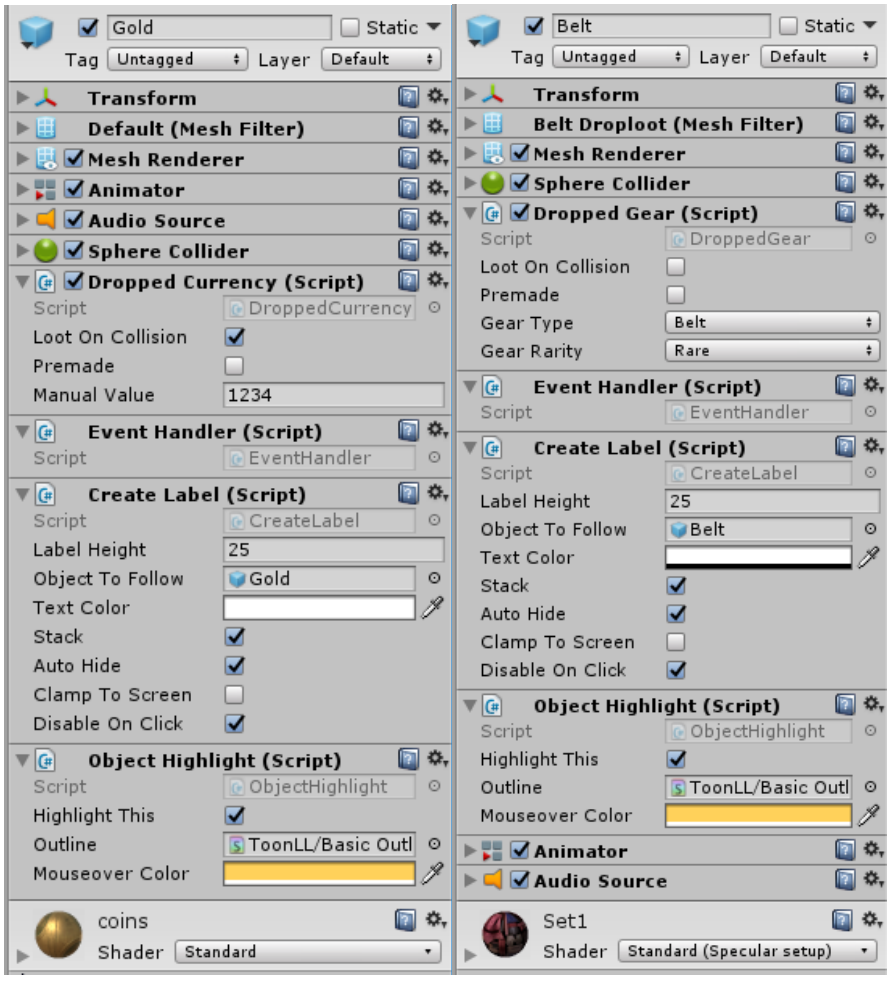
If you want to add/change loot

1) Drag the model for your item in the scene
2) Prepare your object (See previous chapter)
    a. Make sure **premade is turned off** in the DroppedGear/currency script
    b. Add a animator component and assign the dropLoot controller (depending on the scale of your items you might have to adjust the curves of the animation)
    c. Add audio source, turn off play on awake and assign the preferred sound when dropping on the floor
3) Drag your prefab in the Resources/LootLabels/3D Models folder and name it accordingly and remove it from the scene
4) Open the ProjectEnums script (Scripts/Other folder) and add your new item name to the correct array or use visual studio to rename an existing one.
5) Open the resource manager script (Scripts/Managers)
    a. Add a new case to the GetModelName functions with the name you gave to your prefab earlier
    b. Add a new case to the GetIconName functions and fill in the correct icon name from the resources folder
6) Your new item should now drop from chests

# Settings & Script info

## NPC, Container, Loot prefab



### NPC/Container script

This script contains your npc/ container logic, mouse functions and the function to create the label.

- **Object Name**: The name that gets shown in the label
- **Container:**
    - **Loot Source**: Example for container types, used for determining loot drops

### Dropped Gear/Currency script

These scripts contain your gear/currency logic and mouse functions.

- **Loot On Collision**: Loots the item by walking over it with your player
- **Premade**: Normally the dropped item gets generated according to its loot source, but if you just want to put the item in the scene right away, enable premade so the item gets created on awake. Used for debugging or manually adding an object.
- **Gear:**
    - **Gear Type**: Manually chose a gear type for your premade item
    - **Gear Rarity**: Manually chose a rarity for your premade item
- **Currency:**
    - **Manual Value**: Same as dropped gear, but gives a value to your currency

### Create Label script

Creates a label based on the settings provided.

- **Label Height**: The amount of offset the label hovers above the object it follows
- **Object To Follow**: The object you want the label to follow
- **Text Color**: The color of the text (only used if label is spawned by color)
- **Stack**: If checked, the label stacks on other stack labels, if not it just hover above the object and ignores all overlap.
- **Auto Hide Label**: If checked, the label fades away after X seconds and stays hidden until you mouse over the object or push the Show All Loot button
- **Clamp To Screen**: If checked, the label stays on the screen (can be used for important items)
- **Disable On Click**: If checked, the label gets disabled and pooled (F.e. when opening a chest, the label doesn't need to stay active)

### Event Handler

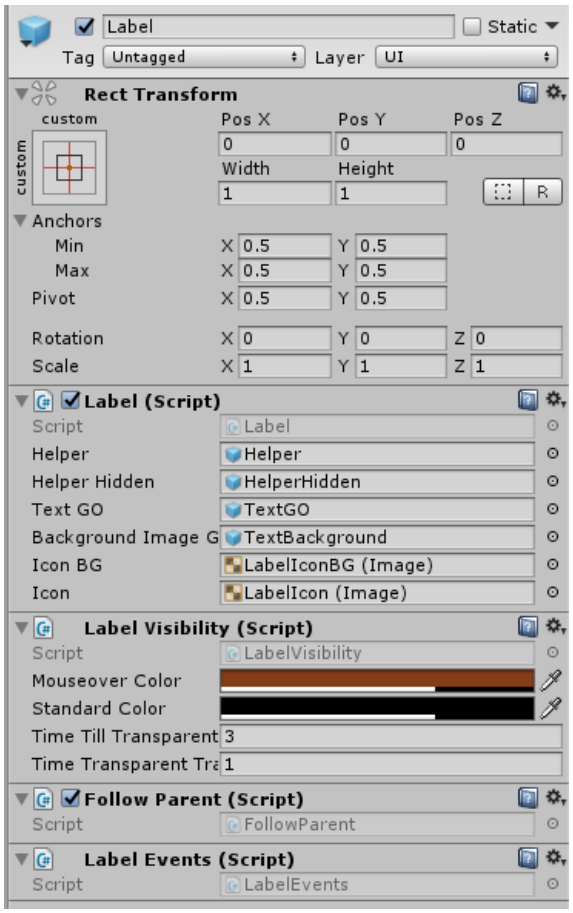
This script contains the mouse and the visibility event raisers. The label's eventhandler keeps a reference of this script so it can call the same function when raising events on the label instead of the 3d object.

### Object Highlight/Mesh Highlight script

The script takes care of highlighting the object when mousing over.
Mesh highlight is only used on the children of your 3d object that need highlighting.

- **Highlight This**: If your parent transform contains a renderer you want highlighted when you mouse over it, check this.
- **Outline**: Assign the shader you want to use for highlighting here. (Toon basic outline)
- **Mouseover Color**: The color of the outline. If you use a different shader you probably need to change the way the color gets assigned to the shader in the script.

## Label prefab




### Label script

This script initializes the label's components and handles state changes.

### Follow Parent Script

This script calculates the worldToScreenPoint for the object it follows.
Both helpers use this value for following, so instead of calculating it twice in each helper we do it once in the parent for performance.

### Label Visibility script

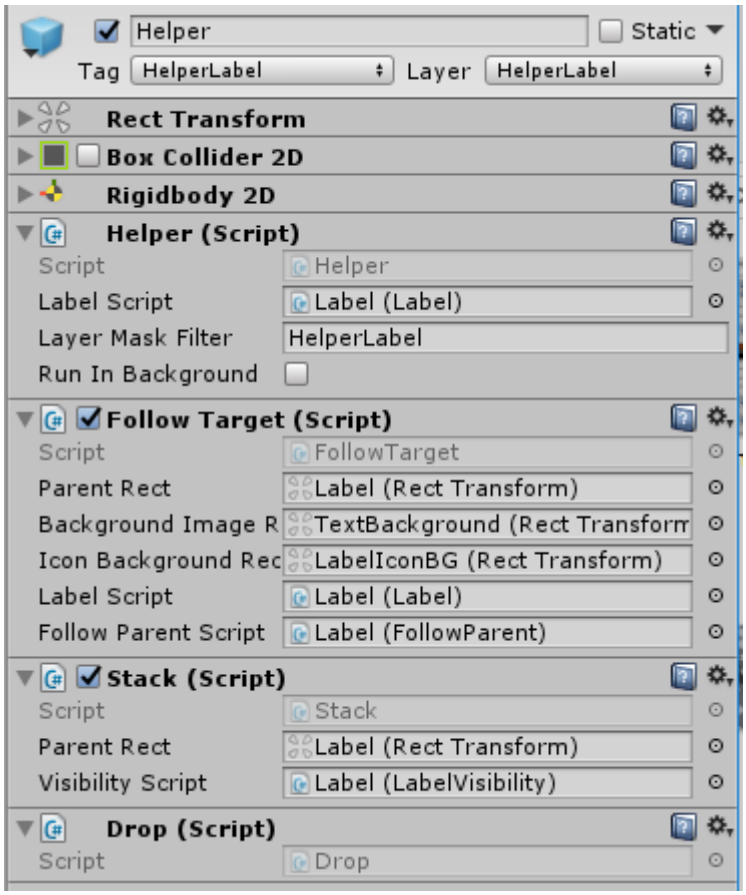
Script containing the logic to show and hide the label

- **Mouseover Color**: The color of the background Image of the text when mousing over
- **Standard Color**: The standard color of the background Image of the text
- **TimeTillTransparent**: When Autohidelabel is turned on for the object, after X seconds the label will start to fade away
- **TimeTransparentTransition**: The time it takes for the label to go from opaque to transparent

### Label Events script

This Script contains the event raiser for the label and the mouse functions to change the label states. It also keeps a reference to the eventhandler from the object it follows and subscribes the label's mouse functions to it so we can call the both the label and the 3D object's mouse function in one call.

## Label Helpers




### Helper script

This script initializes the helper components and handles state changes.

- **Label Script**: Reference to parent label script
- **Layer Mask Filter**: Used by the stack and drop script to boxcast against labels on the correct layer
- **Run in Background**: The helperhidden object stacks “in the background” with other helperhidden objects. They need less functionality then the helper object so instead of making a separate script I just added a bool with separate functions in the same script.

### Stack Script

This script takes care of checking for overlap and assigning the correct target to follow when stacking.

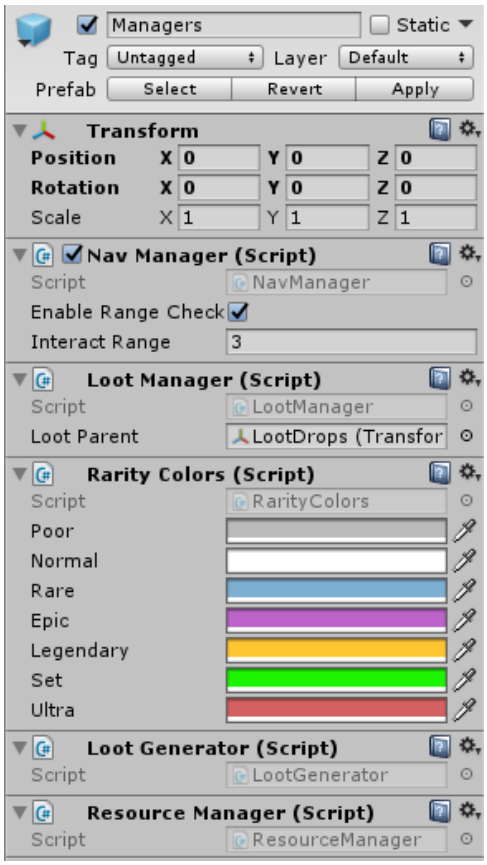
### Drop Script

This script contains the functionality for dropping the labels or resetting them back to their base position.

### Follow Target Script

This script contains the follow logic and calls the drop functions from the drop script.

## Manager object




### Nav Manager Script

The nav manager takes care of moving the nav agent and calling the queued loot/interaction events when out of range.

- **Enable Range Check**: If enabled the script checks if the player is in range of the interaction event
- **Interact Range**: The maximum distance the player needs to be near the object for the interaction to run

### Loot Manager Script

The loot manager takes care of instantiating the correct loot prefabs and assigns the generated stats to the object.

- **Loot Parent**: All the dropped loot gets placed in this transform.

### Rarity Colors script

You can assign a color for each rarity you want. Dropped gear uses this to get the color for its text.

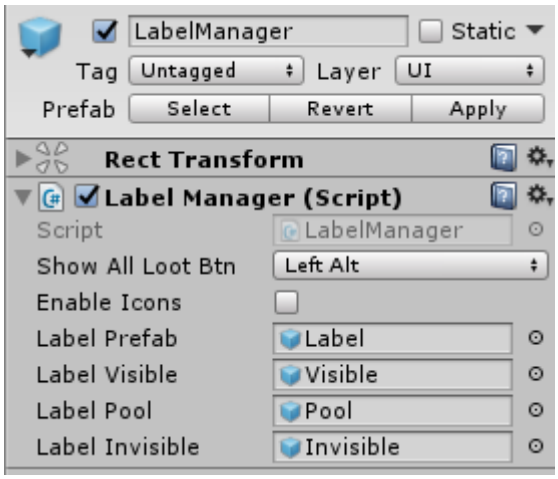
### Loot Generator script

This script contains a couple of example loot generation functions to test different kinds of loot drops. You can expand upon this if you want.

### Resource Manager Script

This script contains switch functions to return the correct path values for object in the resources folder based on the given item values.

## Label Manager




### Label Manager Script

This script is a static singleton that takes care of spawning the labels.

- **Show All Loot Btn**: The button you push to make all labels visible on the screen
- **Enable Icons**: When enabled the labels show icons instead of text
- **Label Prefab**: The label object in the prefabs folder
- **Label Visible**: All labels currently in the camera frustum get placed in this transform
- **Label Pool**: All disabled labels get placed in the pool so they can be reused.
- **Label Invisible**: All labels not in the camera frustum get placed in this transform so they don't get called in the manual update script from Label Visible