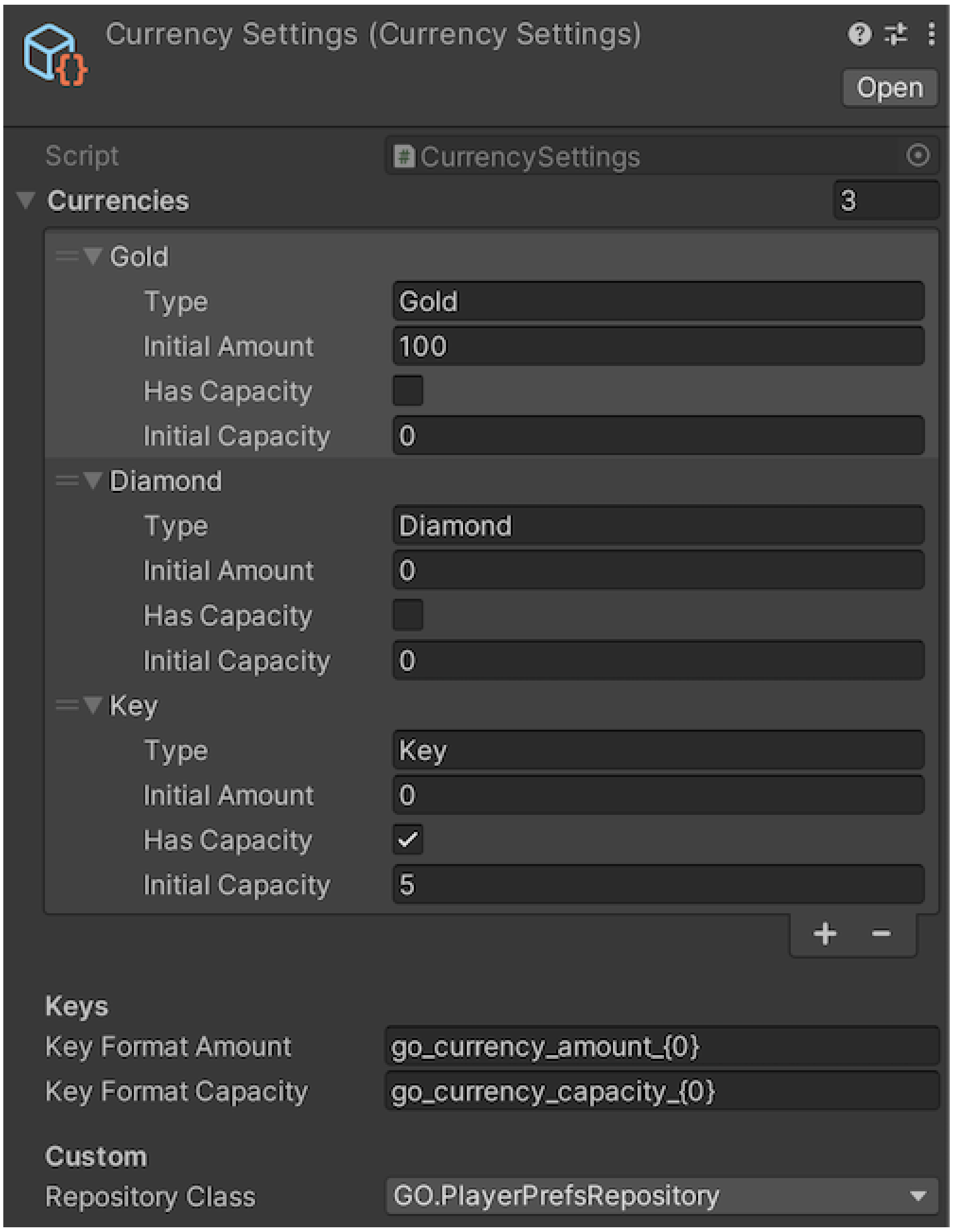
Currency System

Settings

System will automatically generate `CurrencySettings` scriptable object under `Assets/Resources`. Do not move it. Even if it is moved, system will generate a new one there and use it.


Currency Settings (Currency Settings)

Open

Script	CurrencySettings
Currencies	3
Gold	
Type	Gold
Initial Amount	100
Has Capacity	☐
Initial Capacity	0
Diamond	
Type	Diamond
Initial Amount	0
Has Capacity	☐
Initial Capacity	0
Key	
Type	Key
Initial Amount	0
Has Capacity	✓
Initial Capacity	5
Keys	
Key Format Amount	go_currency_amount_{0}
Key Format Capacity	go_currency_capacity_{0}
Custom	
Repository Class	GO.PlayerPrefsRepository

1. Currencies

Here you need to define your currencies.

- `Type` -> Defines the type of currency.
- `Initial Amount` -> With how much currency user will have at start.
- `Has Capacity` -> Capacity option for currencies like key or energy.
- `Initial Capacity` -> Initial capacity of currency.

2. Keys

These keys are used to store currencies' data. By default, it doesn't need a configuration and data is stored at `PlayerPrefs`. If any other method is needed, you need to implement an IRepository that suits your needs. Only then you may need to change these keys to match your services' ids or keys.

3. Custom

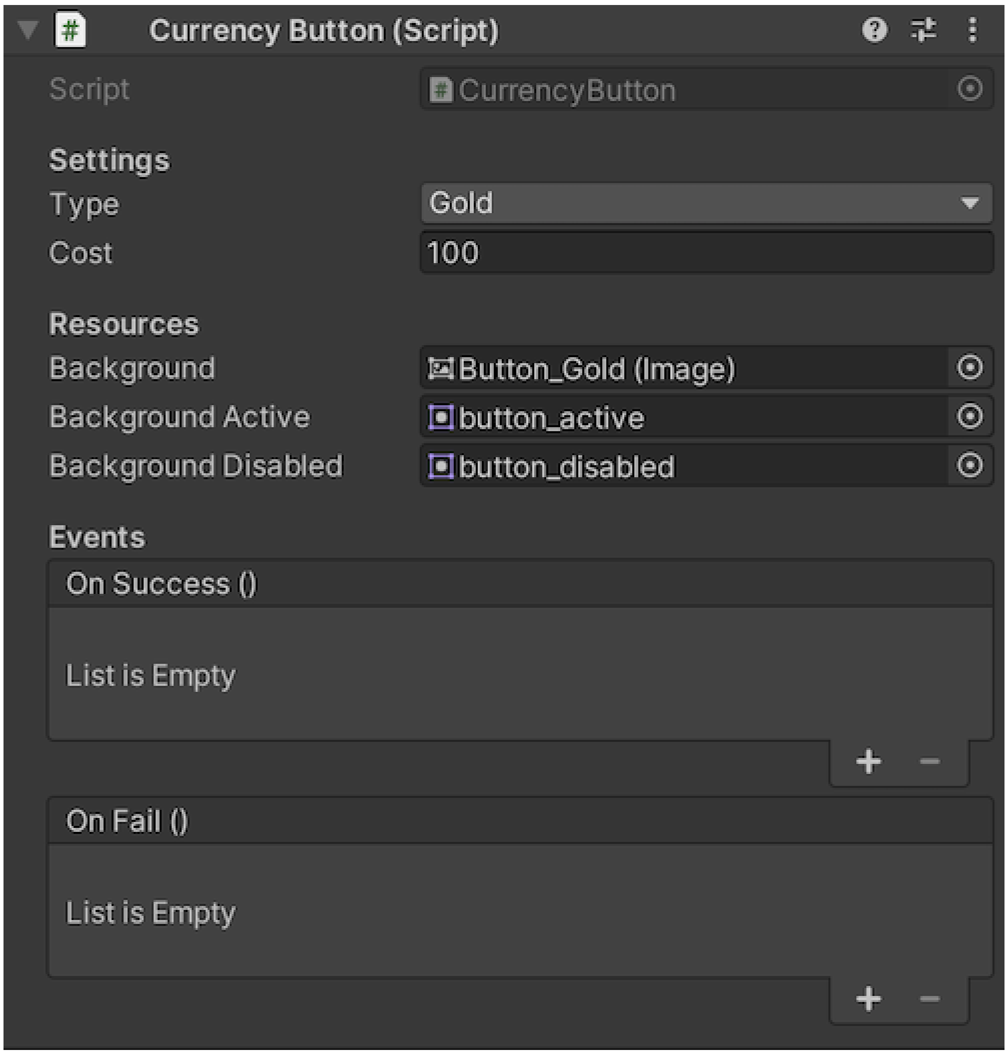
- `Repository Class` -> Type of repository to use.

API Usage

```
// To get currency
int goldAmount = CurrencyManager.Get("Gold");

// To set currency
CurrencyManager.Set("Gold", 100);

// To add currency
int newTotalAfterAddition = CurrencyManager.Add("Gold", 100);

// To remove currency
int newTotalAfterRemoval = CurrencyManager.Remove("Gold", 100);

// To check if has enough
bool hasEnough = CurrencyManager.Has("Gold", 100);

// To delete key data
CurrencyManager.Clear("Gold");

// To get capacity
int keyCapacity = CurrencyManager.GetCapacity("Key");

// To set capacity
CurrencyManager.SetCapacity("Key", 10);

// To check if currency has capacity option
bool hasGoldCapacity = CurrencyManager.HasCapacity("Gold"); // false
bool hasKeyCapacity = CurrencyManager.HasCapacity("Key"); // true


// Events
// Triggered on any currency amount change
CurrencyManager.OnAmountChanged += (type, currentAmount, changeAmount) =>
{
    // Do something
};

// Triggered on any currency capacity change
CurrencyManager.OnCapacityChanged += (type, capacity) => {
    // Do something
};
// Don't forget to unsubscribe
```

Components

1. Currency Button



Currency Button (Script)

Script	CurrencyButton
Settings	
Type	Gold
Cost	100
Resources	
Background	Button_Gold (Image)
Background Active	button_active
Background Disabled	button_disabled

Events

On Success ()

List is Empty

On Fail ()

List is Empty

Its state is changed automatically on currency amounts change.

Settings:

- `Type` -> type of currency
- `Cost` -> cost of transaction

Resources:

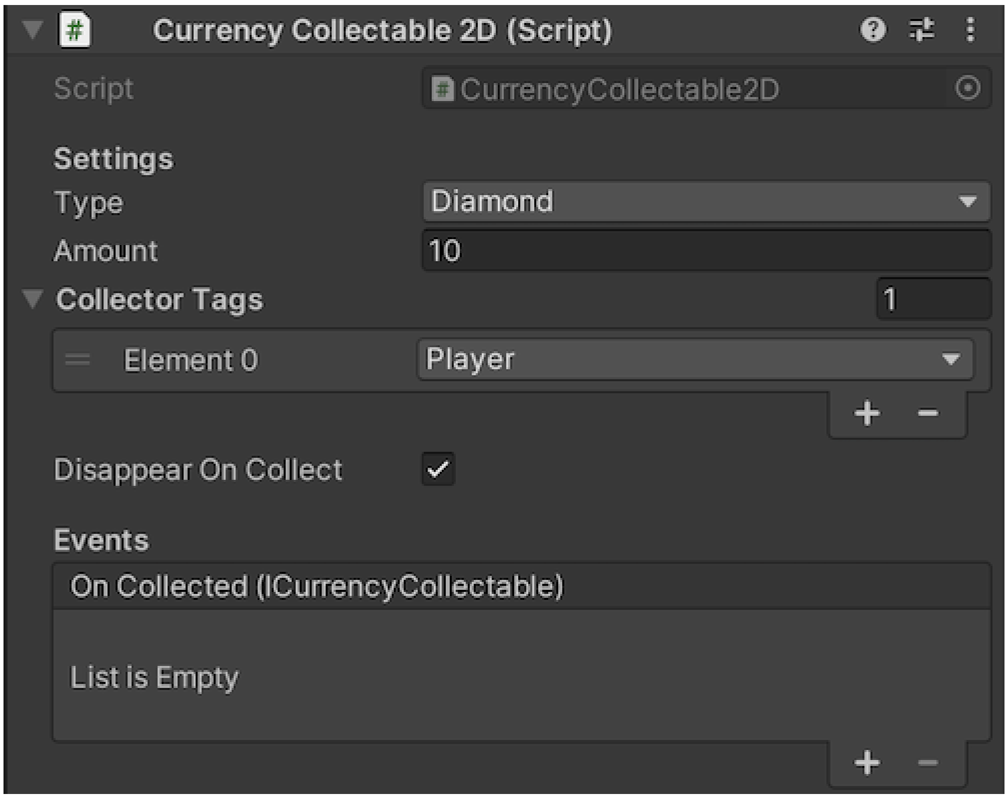
- `Background` -> the reference of background of the button
- `Background Active` -> sprite to set on active state
- `Background Disabled` -> sprite to set on disabled state

Events:

- `OnSuccess` -> Triggered on transaction succeeded

- `OnFail` -> Triggered on transaction failed

2. Currency Collectable



Script: CurrencyCollectable2D

Settings

Type: Diamond

Amount: 10

Collector Tags: 1

Element 0: Player

Disappear On Collect: ✓

Events

On Collected (ICurrencyCollectable)

List is Empty

3D version is identical to 2D. The only difference is the collider type it needs.

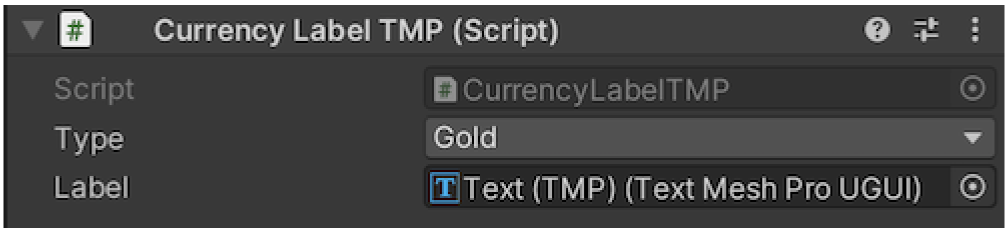
Settings:

- `Type` -> type of currency
- `Amount` -> amount to add on collected
- `Collector Tags` -> allowed tags to collect
- `Disappear On Collect` -> disables game object on collect

Events:

- `On Collected` -> Triggered on collected

3. Currency Label



Script: CurrencyLabelTMP

Type: Gold

Label: Text (TMP) (Text Mesh Pro UGUI)

There are two ready implementation for;

- Unity Text -> `CurrencyLabelText`
- TextMeshPro -> `CurrencyLabelTMP`

New ones can be implemented via `ICurrencyLabel`.

Settings:

- `Type` -> type of currency
- `Label` -> label to show amount

Implementing Custom Repository

You just need to implement this simple interface and change Repository Class to it at the settings.

```
namespace GO
{
    public interface IRepository
    {
        void Set(string type, int value);
        int Get(string type, int defaultValue);
        void Clear(string type);
    }
}
```