Guide d'utilisation du système SecuredValue : valeur sécurisée

Le système `SecuredValue<T>` est conçu pour vous permettre de gérer des valeurs de manière sécurisée dans vos projets Unity. Ce guide vous expliquera comment utiliser ce système correctement.

Introduction

Le système `SecuredValue<T>` vous permet de gérer une valeur de manière sécurisée en la protégeant contre les modifications non autorisées par d'autres objets de votre projet Unity.

Utilisation de SecuredValue<T>

Pour utiliser `SecuredValue<T>`, suivez les étapes suivantes :

1. **Créer une instance de SecuredValue<T>** :
   csharp

```
// Exemple : Création d'une SecuredValue<int> avec une valeur par défaut de 10
SecuredValue<int> mySecuredValue = new SecuredValue<int>(10);
```

Ajouter des modificateurs : Vous pouvez ajouter des modificateurs à votre `SecuredValue<T>` en créant des instances de `SecuredValueAdder<T>` et en les ajoutant à la SecuredValue :
csharp

```
// Exemple : Ajout d'un modificateur avec une priorité de 1 et une valeur de 5
SecuredValueAdder<int> modifier = new SecuredValueAdder<int>(1, 5);
mySecuredValue.AddModifier(modifier);
```

Obtenir la valeur sécurisée : Vous pouvez récupérer la valeur sécurisée en utilisant la méthode `GetValue()` de `SecuredValue<T>` :
csharp

```
// Exemple : Obtention de la valeur sécurisée
int value = mySecuredValue.GetValue();
```

Supprimer des modificateurs : Vous pouvez également supprimer des modificateurs de votre `SecuredValue<T>` si nécessaire :
csharp
```
// Exemple : Suppression du modificateur précédemment ajouté
mySecuredValue.RemoveModifier(modifier);
```

Gérer les événements de valeur modifiée : Vous pouvez vous abonner à l'événement `OnValueChanged` pour être notifié lorsque la valeur sécurisée change :
csharp
```
// Exemple : Abonnement à l'événement OnValueChanged
mySecuredValue.OnValueChanged += newValue => Debug.Log($"New value: {newValue}");
```