Details

Package:	**Bitwise AI Blackboard**
Version:	**1.0.0**
Min Unity version:	**2019.4.0f1 (LTS)**
Author:	**Steven Dalton, Bitwise AI Limited**
Contact:	**steven@bitwiseai.co.uk**

Contents

1.1 Introduction

The Blackboard is a pattern commonly adopted in AI to provide a single, central location to store various data belonging to an NPC or system to be shared between various components.

The Bitwise AI Blackboard provides this data store for any value or reference type keyed against a hash code that is generated from a unique user specified key (string). It provides all expected functionality with constant time lookup and zero garbage generation alongside an editor tool allowing users to view and edit all data within a Blackboard.

1.2 Architecture

All values of a specific type are stored within a single List in the Blackboard. References to these lists are stored within a parent list as IList types, effectively giving the following structure:

```
List<IList>
{
      List<int>,
      List<GameObject>,
      List<Vector3>,
      ...
}
```

Two Dictionaries are maintained within the Blackboard that provide:

- The index of the list containing each type, keyed by Type, and
- The index of the Blackboard data within the typed list, keyed by BlackboardKey Hash.

In order to provide constant time access to the Blackboard it is recommended to create and retain a BlackboardIndex type for every BlackboardKey used. This BlackboardIndex uses the following data in order to provide constant time access to the nested Lists within the Blackboard:

- An index into the List of Lists, and
- An index into the typed list.

2 Usage

2.1 Setting up a Blackboard

Adding a Blackboard to a project can be easily done in 2 ways:

1. Add the provided BlackboardComponent MonoBehaviour to your existing GameObject, or
2. Add a Blackboard directly as a member of an existing class.

2.2 Using a Blackboard

In order to access the Blackboard you will need to add and expose BlackboardKeys wherever they are to be used, for example:

```
[SerializeField] private BlackboardKey m_HealthBBKey
     = new BlackboardKey("Health");
```

This key's string can then be edited to represent what is going to be stored against it in the Blackboard.

To make use of the constant time access feature you will also need to create a BlackboardIndex using your BlackboardKey and Blackboard:

```
// declare BlackboardIndex as a class member
private BlackboardIndex m_HealthBBIndex = default;

...

// create BlackboardIndex for quick access
m_HealthBBIndex
     = m_Blackboard.CreateAccessor<float>(m_HealthBBKey);
```

Once you have a BlackboardIndex you can use it to set and retrieve values from the Blackboard:

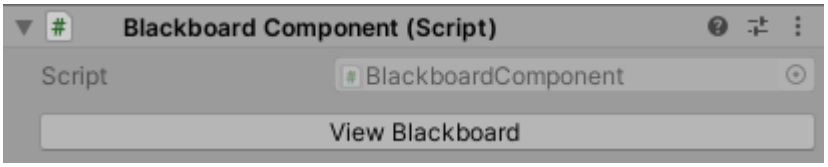
```
// get:
var health = m_Blackboard.QuickGet<float>(m_HealthBBIndex);
// set:
m_Blackboard.QuickSet(m_HealthBBIndex, 100.0f);
```

This is also possible using just the BlackboardKey, although the constant time access is lost :

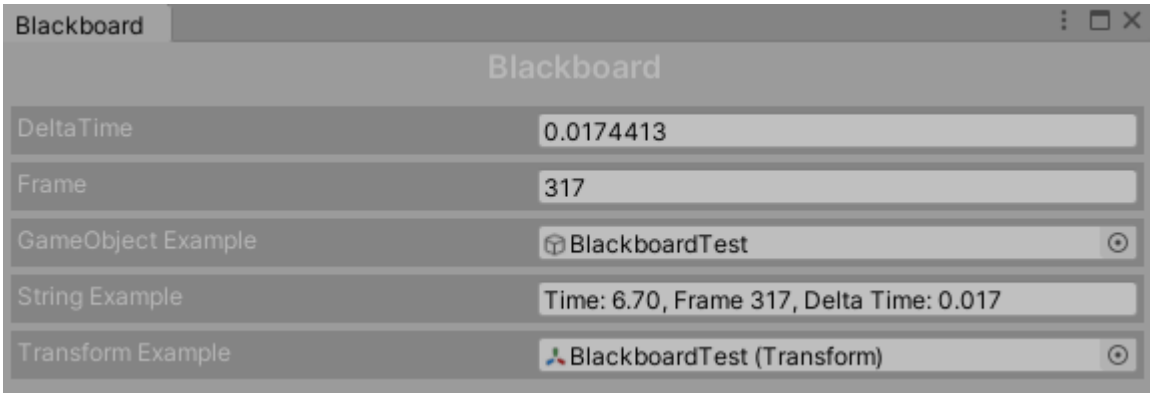
```
// get:
var health = m_Blackboard.Get<float>(m_HealthBBKey);
// set:
m_Blackboard.Set(m_HealthBBKey, 100.0f);
```

2.3 Using the Blackboard Viewer tool

An Editor tool is included to allow users to view and edit all data contained within a Blackboard. The BlackboardComponent MonoBehaviour has a button that will open up the Blackboard Editor window:



Clicking this button will open the Blackboard Editor window for the Blackboard owned by this GameObject.



If you opt not to use the BlackboardComponent MonoBehaviour you should instead call `BlackboardEditor.Open`, passing through the Blackboard that you wish to view. Remember to wrap this call in `#if UNITY_EDITOR`

Editing values in the Blackboard is as simple as clicking and typing for value types, or dragging and dropping new reference types. The following types are supported by default:

- Bool,
- Byte,
- Char,
- Short,
- UShort,
- Int,
- UInt,
- Long,
- Float,
- Enum,
- String,
- Color,
- GameObject,
- Transform,
- Vector2,
- Vector2Int,
- Vector3,
- Vector3Int,
- Vector4,
- BlackboardKey

Adding support for your own types is as simple as creating a new inspector script. This script should be decorated with the following attribute, containing the type you are adding support for:

```
// example that adds support for UnityEngine's GameObject
[BlackboardPropertyInspector(typeof(UnityEngine.GameObject))]
```

This inspector then needs to implement a single static method that takes and returns an object instance. The method should make full use of the available EditorGUILayout serialisation functionality, for example:

```
// allows GameObjects to be viewed in the Blackboard
public static object Inspect(object obj)
{
     return EditorGUILayout.ObjectField(
               (UnityEngine.Object)obj,
               obj.GetType(),
               true);
}
```

2.4 Examples

The following examples can be found in the source:

- Creating a Blackboard:
  - *Assets/BitwiseAI/Blackboard/Scripts/BlackboardComponent.cs - line 11.*
- Creating and exposing a BlackboardKey:
  - *Assets/BitwiseAI/Blackboard/Scripts/Sample/BlackboardSampleComponent.cs - line 7*
- Creating a BlackboardIndex:
  - *Assets/BitwiseAI/Blackboard/Scripts/Sample/BlackboardSampleComponent.cs - line 37*
- Accessing a Blackboard using a BlackboardIndex:
  - *Assets/BitwiseAI/Blackboard/Scripts/Sample/BlackboardSampleComponent.cs - line 56*
- Accessing a Blackboard using a BlackboardKey:
  - *Assets/BitwiseAI/Blackboard/Scripts/Sample/BlackboardSampleComponent.cs - line 62*
- Creating a Blackboard Inspector for an unsupported type:
  - *Assets/BitwiseAI/Blackboard/Scripts/Editor/*Inspector.cs*

If you have any questions, ideas for improving this package or requests for other AI tools please get in touch with me at: **steven@bitwiseai.co.uk**