Hello!

Thank you for downloading the 2D Essential Utils package. I hope this package jumpstarts your journey in 2D game development. The code for this package is heavily commented on, so this document will act as a high level overview of the package features and why you want to use them.

Features:

- Example Scene to demonstrate how the event system and Global Persistent Values can be implemented
- An Event System that supports void and various parameter signatures
- Global Persistent Values stored outside of your scenes which can also be reset on scene load
- Global Persistent Lists stored outside of your scenes, which can be reset on scene load
- Unity Utility methods that I commonly use in my 2D projects
- Unity Extension methods to make working with a 3D-oriented engine easier for 2D developers

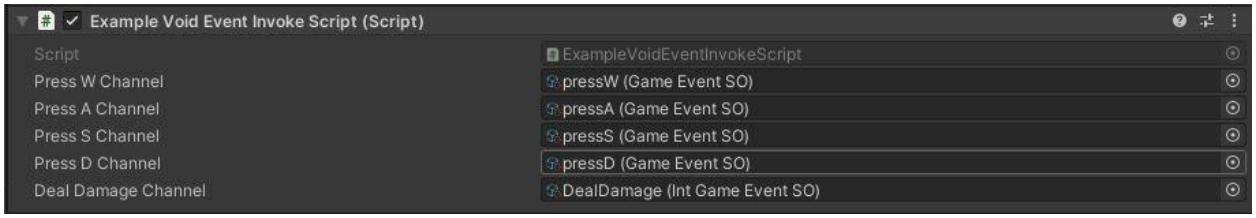
Event System:

The event system allows you to invoke Unity events without requiring a listener. This is useful for decoupling code throughout the project and makes the project more modular. The event system permits sending void events and events with data that the receivers can use in their response.

To create an event:

1. Right click on the projects window and in the drop down go to: Create > Events and select a normal game event of one with a parameter value.
2. Name the event
3. Serialize the event to any scripts that will invoke it
4. Drag the event into the reference field of the object
5. Invoke the event through code

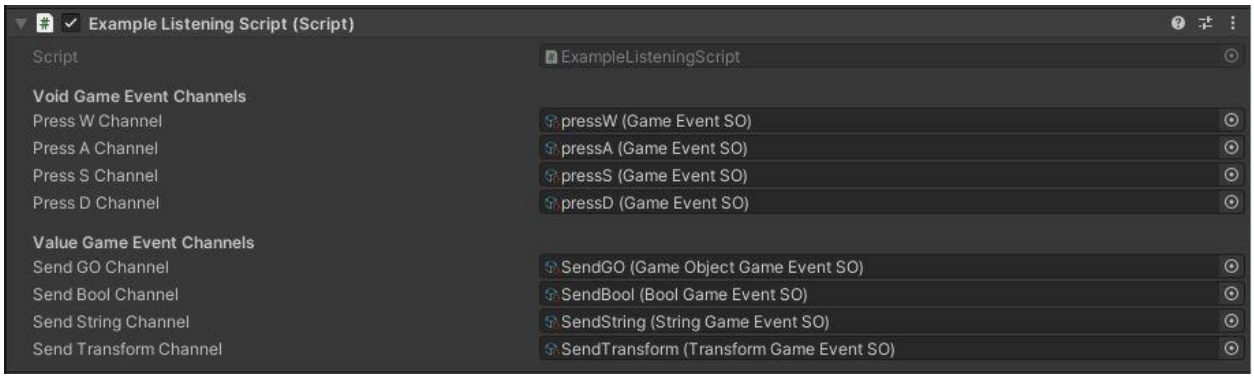
    In my example, the Square Event Invoker, invokes the following events:




```
using UnityEngine;

namespace Nevelson.Utils
{
    Unity Script (1 asset reference) | 0 references
    public class ExampleVoidEventInvokeScript : MonoBehaviour
    {
        [SerializeField] private GameEventSO pressWChannel;
        [SerializeField] private GameEventSO pressAChannel;
        [SerializeField] private GameEventSO pressSChannel;
        [SerializeField] private GameEventSO pressDChannel;
        [SerializeField] private IntGameEventSO dealDamageChannel;


        Unity Message | 0 references
        void Update()
        {
            if (Input.GetKeyDown(KeyCode.W))
            {
                Debug.Log("Pressing W");
                pressWChannel.RaiseEvent();
            }

            if (Input.GetKeyDown(KeyCode.A))
            {
                Debug.Log("Pressing A");
                pressAChannel.RaiseEvent();
            }

            if (Input.GetKeyDown(KeyCode.S))
            {
                Debug.Log("Pressing S");
                pressSChannel.RaiseEvent();
            }

            if (Input.GetKeyDown(KeyCode.D))
            {
                Debug.Log("Pressing D");
                pressDChannel.RaiseEvent();
            }

            if (Input.GetKeyDown(KeyCode.F))
            {
                Debug.Log("Pressing F");
                dealDamageChannel.RaiseEvent(1);
            }
        }
    }
}
```

6. Serialize event in any script that will listen to the event/
7. Drag the event into the reference field of the object
8. Add a response to the event:

    In my example, the Circle Event Listener responds to the events in the following ways:




```
1   using UnityEngine;
2
3   namespace Nevelson.Utils
4   {
        Unity Script (1 asset reference) | 0 references
5       public class ExampleListeningScript : MonoBehaviour
6       {
7           [Header("Void Game Event Channels")]
8           [SerializeField] private GameEventSO pressWChannel;
9           [SerializeField] private GameEventSO pressAChannel;
10          [SerializeField] private GameEventSO pressSChannel;
11          [SerializeField] private GameEventSO pressDChannel;
12
13          [Header("Value Game Event Channels")]
14          [SerializeField] private GameObjectGameEventSO sendGOChannel;
15          [SerializeField] private BoolGameEventSO sendBoolChannel;
16          [SerializeField] private StringGameEventSO sendStringChannel;
17          [SerializeField] private TransformGameEventSO sendTransformChannel;
18
            Unity Message | 0 references
19          private void OnEnable()
20          {
21              pressWChannel.OnEventRaised += MoveUp;
22              pressAChannel.OnEventRaised += MoveLeft;
23              pressSChannel.OnEventRaised += MoveDown;
24              pressDChannel.OnEventRaised += MoveRight;
25
26              sendGOChannel.OnEventRaised += RecieveGO;
27              sendBoolChannel.OnEventRaised += RecieveBool;
28              sendStringChannel.OnEventRaised += RecieveString;
29              sendTransformChannel.OnEventRaised += RecieveTransform;
30          }
31
            Unity Message | 0 references
32          private void OnDisable()
33          {
34              pressWChannel.OnEventRaised -= MoveUp;
35              pressAChannel.OnEventRaised -= MoveLeft;
36              pressSChannel.OnEventRaised -= MoveDown;
37              pressDChannel.OnEventRaised -= MoveRight;
38
39              sendGOChannel.OnEventRaised -= RecieveGO;
40              sendBoolChannel.OnEventRaised -= RecieveBool;
41              sendStringChannel.OnEventRaised -= RecieveString;
42              sendTransformChannel.OnEventRaised -= RecieveTransform;
43          }
```

```
44
                2 references
45              private void MoveUp()
46              {
47                  Debug.Log("Heard W, Moving up!");
48                  transform.Position2D(transform.Position2D() + Vector2.up);
49              }
50
                2 references
51              private void MoveLeft()
52              {
53                  Debug.Log("Heard A, Moving Left!");
54                  transform.Position2D(transform.Position2D() + Vector2.left);
55              }
56
                2 references
57              private void MoveDown()
58              {
59                  Debug.Log("Heard S, Moving Down!");
60                  transform.Position2D(transform.Position2D() + Vector2.down);
61              }
62
                2 references
63              private void MoveRight()
64              {
65                  Debug.Log("Heard D, Moving Right!");
66                  transform.Position2D(transform.Position2D() + Vector2.right);
67              }
68
                2 references
69              private void RecieveGO(GameObject value)
70              {
71                  Debug.Log($"Recieved GO: {value.name}");
72              }
73
                2 references
74              private void RecieveBool(bool value)
75              {
76                  Debug.Log($"Recieved bool: {value}");
77              }
78
                2 references
79              private void RecieveString(string value)
80              {
81                  Debug.Log($"Recieved string: {value}");
82              }
83
                2 references
84              private void RecieveTransform(Transform value)
85              {
86                  Debug.Log($"Recieved transform: {value.name}");
87              }
88          }
89      }
```

9. Now when you run the game, whenever the event is invoked, the listener will respond.

Global Persistent Values And Lists:

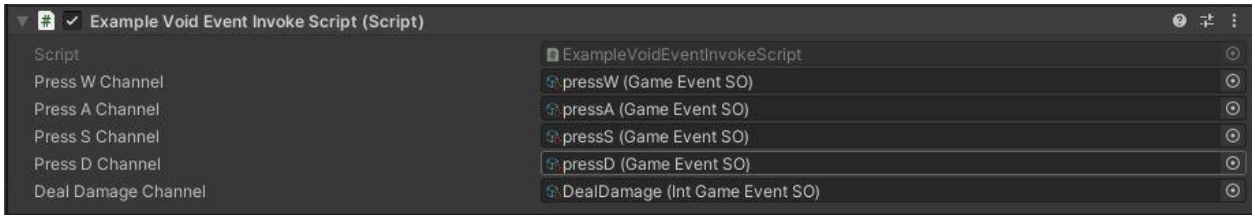
Global Persistent Values and Lists allow you to store globally accessible data outside of your scene and lets you access it without the need of In-game reference objects like game managers. This helps by reducing the dependencies between game objects and global data because a common issue with singletons and monosingleton patterns is that they form dependency chains which lead to NPEs.

The Global Persistent values and lists are only referenced by the objects that need them, and do not care if they are referenced between scenes. They also come with a handy Reset On

Scene Load feature that lets you reset them to preselected default values whenever a new scene loads.

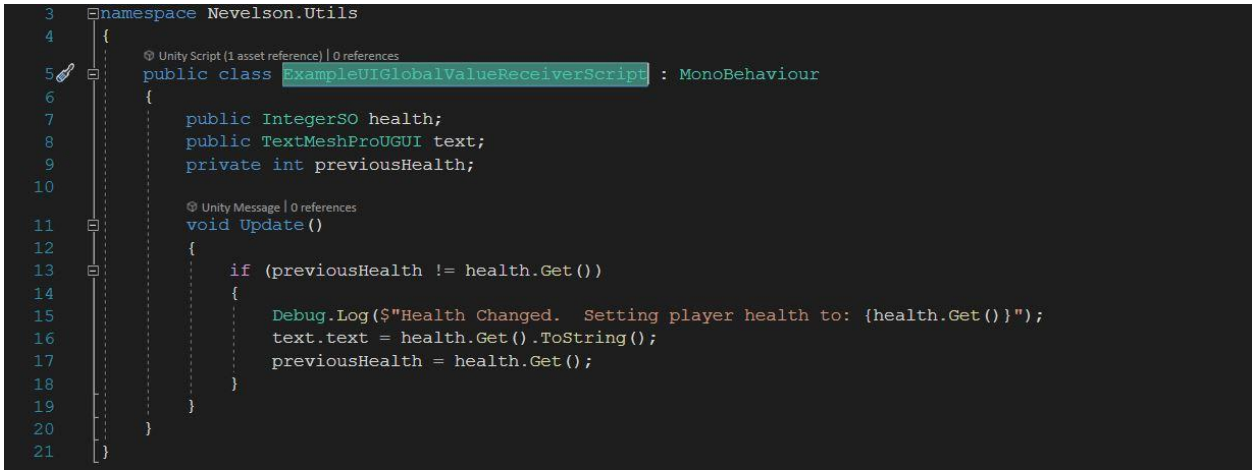
To create a Global Persistent Value:

1. Create a Resources Folder in your project assets. Unity needs this folder to load any data that exists outside of the scene.
2. Inside your Resources Folder, right click > Create > Global Values > and select the type of value or list you want to create.
3. Name your Persistent Value. I called mine Player Health do demonstrate a use case
4. Select whether you would like the Reset On Load to be:
   -Persistent, never resets on its own, even when you exit the game.
   -Reset, resets the value to the supplied default when a new scene loads
5. Serialize the event to any scripts that will invoke it
6. Drag the event into the reference field of the object
   In my example, the Square Event Invoker has a reference to this damage and updates the damage value when the player presses F:



```
using UnityEngine;

namespace Nevelson.Utils
{
    Unity Script (1 asset reference) | 0 references
    public class ExampleVoidEventInvokeScript : MonoBehaviour
    {
        [SerializeField] private GameEventSO pressWChannel;
        [SerializeField] private GameEventSO pressAChannel;
        [SerializeField] private GameEventSO pressSChannel;
        [SerializeField] private GameEventSO pressDChannel;
        [SerializeField] private IntGameEventSO dealDamageChannel;


        Unity Message | 0 references
        void Update()
        {
            if (Input.GetKeyDown(KeyCode.W))
            {
                Debug.Log("Pressing W");
                pressWChannel.RaiseEvent();
            }

            if (Input.GetKeyDown(KeyCode.A))
            {
                Debug.Log("Pressing A");
                pressAChannel.RaiseEvent();
            }

            if (Input.GetKeyDown(KeyCode.S))
            {
                Debug.Log("Pressing S");
                pressSChannel.RaiseEvent();
            }

            if (Input.GetKeyDown(KeyCode.D))
            {
                Debug.Log("Pressing D");
                pressDChannel.RaiseEvent();
            }

            if (Input.GetKeyDown(KeyCode.F))
            {
                Debug.Log("Pressing F");
                dealDamageChannel.RaiseEvent(1);
            }
        }
    }
}
```

In my example, the ReceiverPlayerHealth UI element also has a reference to player health and reads it on update:



```
namespace Nevelson.Utils
{
    Unity Script (1 asset reference) | 0 references
    public class ExampleUIGlobalValueReceiverScript : MonoBehaviour
    {
        public IntegerSO health;
        public TextMeshProUGUI text;
        private int previousHealth;

        Unity Message | 0 references
        void Update()
        {
            if (previousHealth != health.Get())
            {
                Debug.Log($"Health Changed.  Setting player health to: {health.Get()}");
                text.text = health.Get().ToString();
                previousHealth = health.Get();
            }
        }
    }
}
```

Utility Methods:

The following is a list of all utility methods and their summaries.

float ConvertVolumeToLogarithmic(float volume)
Summary: Converts a float from range 0 to 1 to logarithmic scaling, which is what the mixer group uses

Direction GetCardinalDirection(Vector2 startPoint, Vector2 endPoint)
Summary: Returns a cardinal direction based on start and end point. Returns NONE if there is no difference between start and endpoint

string FormatTime(float time)
Summary: Formats seconds to a readable time string: minutes : seconds : milliseconds

string StripPunctuation(string s)
Summary: Strip all punctuation out of a string

T CopyComponent<T>(T original, GameObject destination) where T : Component
Summary: Copies an original component and applies its field values to the copy. Applying fields does not work on compiled getter setters.

List<T> FindComponentsOfType<T>()
Summary: Improved find components of type that permits finding of interfaces as well

bool LerpToDestination(Transform lerpTarget, Vector2 endPos, float speed)
Summary: A continuous lerp to destination. The lerpTarget keeps moving until it reaches the exact point. Does not go beyond.

Transform FindChildWithTag(Transform parent, string tag)
Summary: Finds first child with specified tag using depth first search

TGameObject FindChildWithTag(GameObject parent, string tag)
Summary: Finds first child with specified tag using depth first search

UtilCoroutines Methods:

The following is a list of all coroutine utility methods and their summaries. Please not that to run these they need to be wrapped in a StartCoroutine() function
Ex: StartCoroutine(Utils.WaitForSeconds(action, waitTime));

IEnumerator WaitForSeconds(Action action, float waitTime)
Summary: Waits for seconds before performing an action. This timer slows with timescale.

IEnumerator RealTimeWaitForSeconds(Action action, float waitTime)
Summary: Waits for seconds before performing an action. This timer is not affected by timescale.

IEnumerator WaitForFrames(Action action, int frames = 1)
Summary: Waits for X amount of frames before executing coroutine.

Extensions Methods:

The following is a list of all extension methods, the components they extend and their summaries. To use an extension method, simply call it from the component it is extending.
Ex:
Vector2 myVector = Vector2.zero;
myVector.Rotate(5f);

Extends Vector3

Vector2 Get2D(this Vector3 v)
Summary: Gets the X,Y Vector2 of this Vector3.

Extends Vector2

Vector2 Rotate(this Vector2 v, float degrees)
Summary: Rotates the vector2 by the supplied amount of degrees.

float GetAngle(this Vector2 direction)
Summary: Returns the rotation of a direction.

Quaternion GetRotation(this Vector2 direction)
Summary: Returns the rotation of a direction.

bool TryGetCardinalDirection(this Vector2 start, Vector2 end, out Direction direction)

Summary: Gets the cardinal direction of 2 vectors if they evaluate to an exact cardinal direction. Vector2.Zero returns true with a direction of none.

bool TryGetDirection(this Vector2 start, Vector2 end, out Direction direction)

Summary: Gets the direction of 2 vectors if they evaluate to an exact direction. Vector2.Zero returns true with a direction of NONE.

Vector2 GetDirection(this Vector2 start, Vector2 end)

Summary: Gets the direction from the vector2 to the end position

Vector2 GetRawDirection(this Vector2 start, Vector2 end)

Summary: Gets the raw direction from the vector2 to the end position

Vector2 GetNormalizedDirection(this Vector2 start, Vector2 end)

Summary: Gets the normalized direction from the vector2 to end position.

bool IsAbove(this Vector2 v, Vector2 worldPos)

Summary: Checks if this vector 2 is above the worldPos on the vertical axis.

bool IsAboveOrOn(this Vector2 v, Vector2 worldPos)

Summary: Checks if this vector 2 is above or exactly on the same point of the vertical axis as the world position.

bool IsBelow(this Vector2 v, Vector2 worldPos)

Summary: Checks if this vector 2 is below the world position on the vertical axis.

bool IsBelowOrOn(this Vector2 v, Vector2 worldPos)

Summary: Checks if this vector 2 is below or exactly on the same point of the vertical axis as the world position

bool IsRightOf(this Vector2 v, Vector2 worldPos)

Summary: Checks if this vector 2 is on the right side on the horizontal axis of the world position.

bool IsRightOfOrOn(this Vector2 v, Vector2 worldPos)

Summary: Checks if this vector 2 is on the right side or exactly on the same point of the horizontal axis as the world position

bool IsLeftOf(this Vector2 v, Vector2 worldPos)

Summary: Checks if this vector 2 is on the left side on the horizontal axis of the world position.

bool IsLeftOfOrOn(this Vector2 v, Vector2 worldPos)
Summary: Checks if this vector 2 is on the left side or exactly on the same point of the horizontal axis as the world position

Extends Transform:

void LookAt2D(this Transform t, Transform target)
Summary: Rotates the transform to look at the target world position

void LookAt2D(this Transform t, Vector2 worldPos)
Summary: Rotates the transform to look at the target world position

List<Transform> GetDirectChildren(this Transform t)
Summary: Returns all child transforms for the gameobject

List<Transform> GetAllChildren(this Transform t)
Summary: Returns all child transforms for the gameobject

void DestroyAllChildren(this Transform t, bool destroyImmediately = false)
Summary: Destroys all children of supplied GO transform, by default destroys after 1 frame delay

Vector2 LocalScale2D(this Transform t)
Summary: Gets the local scale of the gameobject in 2D. The Zed axis is assumed to be 1.

void LocalScale2D(this Transform t, Vector2 localScale2D)
Summary: Sets the local scale of the gameobject in 2D. The Zed axis is assumed to be 1.

Vector2 GetNormalizedDirection(this Transform t, Vector2 target)
Summary: Gets the direction from object to target transform.

Vector2 GetNormalizedDirection(this Transform t, Transform target)
Summary: Gets the normalized direction from object to target transform.

Vector2 Position2D(this Transform t)
Summary: Gets the world position of the gameobject in 2D. The Zed axis is assumed to be 0.

void Position2D(this Transform t, Vector2 position2D)
Summary: Sets the world position of the gameobject in 2D. The Zed axis is assumed to be 0.

Vector2 LocalPosition2D(this Transform t)
Summary: Gets the local position of the gameobject in 2D. The Zed axis is assumed to be 0.

void LocalPosition2D(this Transform t, Vector2 localPosition2D)
Summary: Sets the local POSITION of the gameobject in 2D. The Zed axis is assumed to be 0.

Extends SpriteRenderer:

bool OrientZeroPreferRight(this SpriteRenderer spriteRenderer, Vector2 lookAtPoint)
Summary: Flips the sprite if the look at point is to the left of the sprite renderer. If they are equal on the horizontal axis, it will prefer to stay unflipped flip

bool OrientNoDirZeroPreference(this SpriteRenderer spriteRenderer, Vector2 lookAtPoint)
Summary: Flips the sprite if the look at point is to the left of the sprite renderer. If they are equal on the horizontal axis, it will remain in the state that it's currently in.

Extends RigidBody2D:

Vector2 PredictFuturePos(this Rigidbody2D rb, int fixedFramesAhead)
Summary: Predicts future position of the object based on it's current velocity

Vector2 PredictFuturePosFromChildPos(this Rigidbody2D rb, Vector2 position, int fixedFramesAhead)
Summary: Predicts future position of the object based on its current velocity. Used when the position is derived from a child object rather than the rigidbody itself. Ex: Want to use a sub child's position

Extends Renderer:

bool IsVisibleFrom(this Renderer renderer, Camera camera)
Summary: Checks if the renderer component is visible from the camera

Extends RectTransform:

void SetLeft(this RectTransform rt, float left)
Summary: Sets the rect transform X pixels left of anchor

void SetRight(this RectTransform rt, float right)
Summary: Sets the rect transform X pixels right of anchor

void SetTop(this RectTransform rt, float top)
Summary: Sets the rect transform X pixels up of anchor

void SetBottom(this RectTransform rt, float bottom)
Summary: Sets the rect transform X pixels bottom of anchor

Extends Layermask:

bool IsInLayerMask(this LayerMask layerMask, string layerName)
Summary: Checks if the layer name is part of a layermask

Extends GameObject:

void WhileSceneLoaded(this GameObject gameObject, Action action)
Summary: Call this method in the Update, OnDisable, or OnDestroy function to ensure that the action you provide is only called while the scene is loaded. This method prevents the action from happening while a scene is unloading. Useful for avoiding calling runtime specific OnDestroy functions while the scene is being cleaned. Ex. OnDestroy if you call a function like: SetUIScore++, you do not want this happening when the scene is being cleaned up and destroying all objects.

void WhenSceneUnloads(this GameObject gameObject, Action action)
Summary: Call this method in the OnDisable, or OnDestroy function if you only want the action to run while the scene is being cleaned up and unloaded, but not when the component is destroyed by other means. Ex. OnDestroy if you call a function like: ResetPlayerScore(), you do not want this happening if the player dies and respawns without gameover, but you do want to reset the score when the move to the next level.