Lerper / Tweener

namespace - XGI.Tools

Lerping - Tweening is made as simple as possible

How to to Lerp -

```
using UnityEngine;
using XGI.Tools;

public class SampleScript : MonoBehaviour
{
    public GameObject objectToMove;
    public Transform targetPosition;

    private void OnEnable()
    {
        Lerper lerper = objectToMove.AddComponent<Lerper>();
        lerper.MoveObject(targetPosition);
    }
}
```

How to extract the journey progress percentage of the lerped object -

```
float completed = lerper.progress;
```

Replace Lerper with Tweener it everything will still work

Properties

- float progress => Journey progress percentage

Lerper Methods

- void MoveObject(Transform moveToPosition, float Movementspeed = 10, bool destroyGameObjectAfterLerping = false) => Makes the GameObject move

Tweener Methods

- void MoveObject(RectTransform moveToPosition, float timeTaken = 1, bool destroyGameObjectAfterTweening = false) => Makes the UI object move

DataFileManager

namespace - XGI.Tools

This is an encapsulation of all the necessary features required for saving and loading game data. It makes the processes easier and compact.

It is an abstract class, so you need to derive it to the data container class in order to use it. The container class will then have all the features of DataFileManager class.

The process of doing so is as follows -

```
using System;
using XGI.Tools;
⋮
[Serializable]
public class GameData : DataFileManager
{
    ⋮   //Game Data

    public GameData(string directory) : base(directory){}
    public override object GetInstance() { return this; }
}
```

Once the data container class is set up, all that needs to be done, is creating an object of the class.
The save game directory is to passed along with instantiation of this class.

```
public GameData gameData;
private void Awake() // Or anywhere appropriate
{
    gameData = new GameData(Application.persistentDataPath + "data.dat");
}
```

Everything is done, now the game can be saved or loaded any time using only one line of code which would be -

```
gameData.SaveData();
```

And

```
gameData = (GameData)gameData.LoadData();
```

Respectively...

Constructor -

- DataFileManager(string directory) => The directory where the game is to be saved. Usually it should start with Application.persistentDataPath + ...

Properties -

- bool IsDataExist => True if save file exist in the said location, false otherwise

Methods -

- void SaveData() => Save the data
- object LoadData() => Returns boxed saved data, it must be casted to the type of its serializable data container class

MethodHandler

namespace - XGI.Tools

This class enables the user to treat methods like properties i.e. pass them as arguments, copy them, use them as return variable from other methods etc. MethodHandler is much more flexible and mobile as compared to Delegates or Actions.

How to Initialize a method handler -

```
public SampleClass
{
    private void OnEnable()
    {
        MethodHandler demoHandler = new
MethodHandler(typeof(NewBehaviourScript).GetMethod("TestMethod"), this, new object[]{})
    }
⋮
    public string TestMethod() => "Test Method Invoked";
}
```

Now demoHandler can be passed arround like any other variable.

How to Invoke a method stored using MethodHandler -

```
string handlerReturnValue = demoHandler.Execute();
```

Or simply

```
demoHandler.Execute();
```

If no value is to be returned.

Constructor -

- MethodHandler(MethodInfo methodInformation, object instance, object[] arguments)

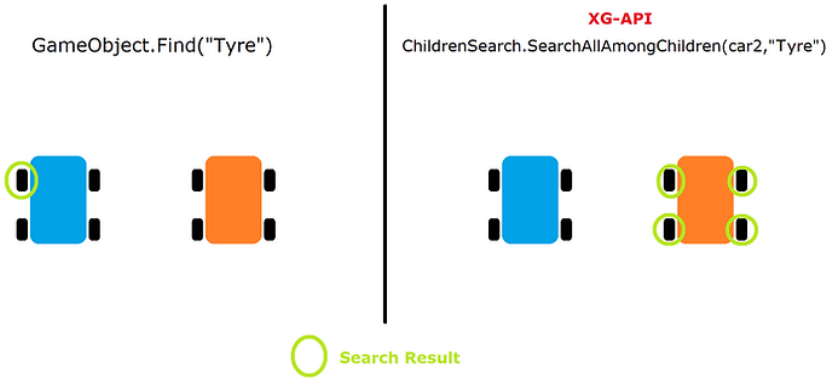
Methods -

- Type GetReturnType() => Returns the return type of the stored method
- object Execute() => Executes the stored method

ChildrenSearch

namespace - XGI.Tools

Children Search works like GameObject.Find() but the user has far more control over the search results. An example of which is demonstrated below.



Methods -

- static GameObject[] SearchAllAmongChildren(GameObject self, string childname, bool caseSensitivity = false) => This Function performs a searchs for all the GameObjects with a specified name, among the children of a GameObject
- static GameObject SearchAmongChildren(GameObject self, string childname, bool caseSensitivity = false) => This Function performs a searchs for a GameObject with a specified name, among the children of a GameObject
- static GameObject[] GetChildren(GameObject self) => This Function performs a searchs for all the children of a GameObject
- static GameObject[] SearchAllAmongLineage(GameObject self, string childname, bool caseSensitivity = false) => This Function performs a searchs for all the GameObjects with a specified name, among the lineage of a GameObject
- static GameObject SearchAmongLineage(GameObject self, string childname, bool caseSensitivity = false) => This Function performs a searchs for a GameObject with a specified name, among the lineage of a GameObject
- static GameObject[] GetLineage(GameObject self) => This Function performs a searchs for the lineage of a GameObject