UnityLibrary - Documentation

https://github.com/Luca00IT/UnityLibrary

Description

UnityLibrary contains many different libraries for Unity to speed up the process of developing your own video game. Later you can consult the documentation in order to understand how to implement these libraries in your projects and use the functions.

Prerequisites

- Unity Engine v2021.3.4f1 or higher
- Visual Studio 2022 Community + "Game Development with Unity" extension
- WinRAR, WinZip, 7zip or any file storage program

How to install

1. Download the library package
2. Extract the package and place it inside the "Assets" folder of your Unity project

FadeTools

This library contains methods for applying a "FadeIn" and "FadeOut" effect to your RawImage, which can be used, for example, when you want to simulate the loading of a layer or for other functions.

Import the libraries

```
using UnityEngine;
using UnityEngine.UI;
using FadeTools;
```

Initialize the variables

```
[SerializeField]
private RawImage rawImage; // this is the image that will appear/disappear

[SerializeField]
private float duration; // this is the time used by the fade functions to terminate
```

IEnumerator FadeIn(rawImage, duration)

```
StartCoroutine(Fade.FadeIn(rawImage, duration);
```

Start this routine to make the rawImage darker and darker in a time determined by the duration variable.

IEnumerator FadeOut(rawImage, duration)

```
StartCoroutine(Fade.FadeOut(rawImage, duration);
```

Start this routine to make the rawImage lighter until they disappear in a time determined by the duration variable.

ActionTools

This library can be used in the field of 3d first-person games to apply an interaction to a GameObject based on the tags.

Import the library

```
using UnityEngine;
using ActionTools;
```

Initialize your variables

```
[SerializeField]
private List<string> interactiveTags = new List<string>; // this is the list of interactive tags

[SerializeField]
private int tagIndex; // this is the location on the list of the interactive tag

[SerializeField]
private float maxRaycastDistance; // this is the distance at which the raycast will work
```

Action.SaveTags(List)

```
void Start()
{
    Action.SaveTags(interactiveTags); // store the tags into a .dat file
}
```

Calling this function will store all the tags you saved in your own list to a *tags.dat* file, located inside your project folder. From now on the GameObjects that will use these tags are all "interactive" and can activate a method written by yourself.

Action.IsObjectInteractable(int, float)

```
void Update()
{
    if (Action.IsObjectInteractable(out tagIndex, maxRaycastDistance))
    {
        switch(tagIndex)
        {
            case 0:
                //do stuff here
                Debug.Log("The GameObject is interactive: " + interactableTags[tagIndex]);
                break;
        }
    }
}
```

This function checks every frame if the GameObject the player is looking is interactive. If the GameObject is interactive then the function can now call a method written by yourself.

NOTE: it is recommended to use the switch and cases for each method

License