TOOL-BELT

Made by Star Banner Games.

Itch.io: Star Banner Games **LinkedIn:** Sanandana Michael Ithaler **Twitter:** @starbannergames

Content

1 Overview

To avoid bloated Projects, Toolbelt is split up into Chunks which (except for "Core") work independently from eachother:

- **Core:** Utility Classes and some basic features that will prove useful in most projects. *Core always needs to be included when using Toolbelt!*
- **AppLinker:** A small Tool that lets you create links to other Applications. This way you can e.g: open a texture in Photoshop directly through Unity or open your recording software to make a quick GIF.
- **DebugTools:** This module includes a variety of Gizmo Classes, as well as Attributes to make your editor experience more efficient and clearer.
- **Optimization:** Simple Shortcuts to increase your Performance
- **SceneSelector:** A Scene Switcher that lets you customize shortcuts to your scenes for a nice overview.
- **UI:** This module contains UI Templates and Managers to create common UI Elements with a nice level of polish.

To use any of these modules, include the namespace **"SBG.Toolbelt.ModuleName".** You can find a more detailed documentation in the code itself, but for clarities sake, the following is a quick introduction to each module!

If this Asset is useful to you, please consider rating it. Thank you!

2 Documentation

2.1 Core

2.1.1 Randomizer

The Randomizer Class contains a bunch of functions to get different random/chance values.

2.1.2 THelper

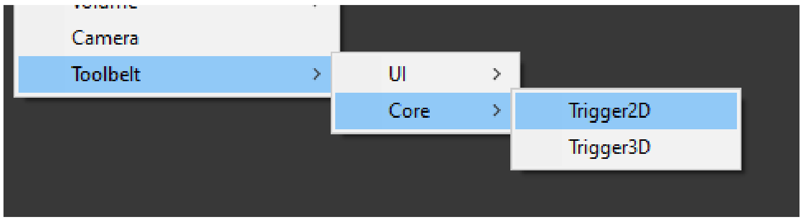
THelper is a Utility class containing all kinds of operations, including some Vector & Quaternion Math.

2.1.3 ValueRange

ValueRange is a datatype used for Min/Max declarations. This is meant to make your fields and your inspector a bit cleaner.

2.1.4 Trigger 2D/3D

Two Trigger Components (for 2D & 3D respectively), that allow you to assign Actions to Trigger/Collision Events. You can spawn a configured Prefab by right-clicking in your Scene Hierarchy and navigating to **"Toolbelt/Core/".**



2.2 AppLinker

To configure AppLinker, navigate to:
"Window/StarBannerGames/Toolbelt/AppLinkerSettings".

There you can add/remove applications and configure them. If an application should not be used to edit assets, but just as a shortcut to the application itself, change the Application Settings from "Default Link" to "Toolbar Link". Please note that applying your settings requires unity to recompile and might take a few seconds.

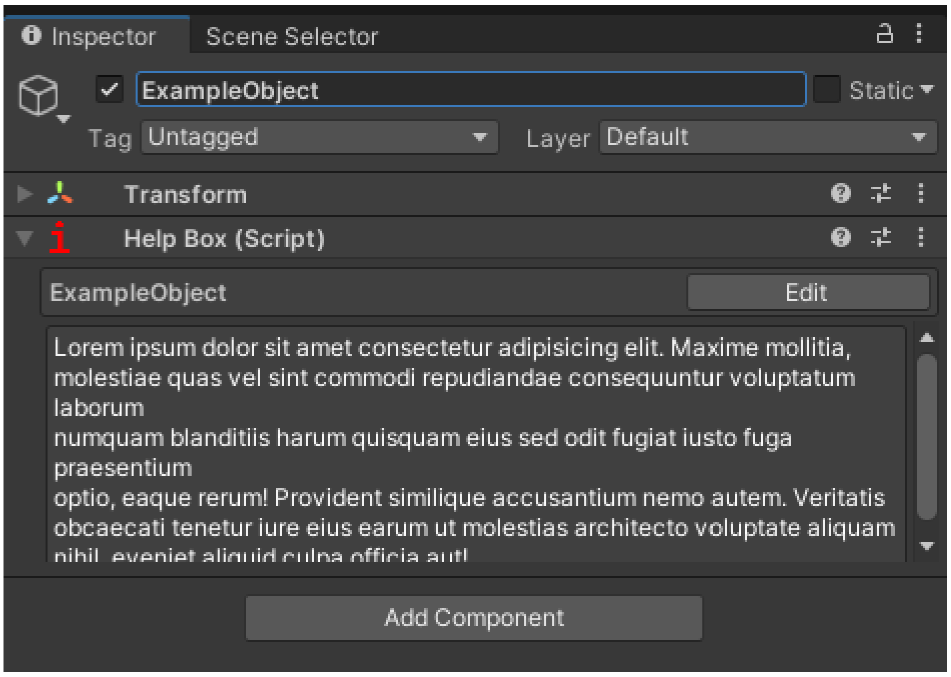
Once your Settings are applied, you have two ways of using AppLinker:

- The Unity Toolbar should now have a new Tab called "App Links", containing all the apps you added. Simply click one to open it.
- Right-Click an Asset and select **"Open In App/[AppName]"**. This wont work on Apps that are set to "Toolbar Link".

2.3 Debug Tools

2.3.1 HelpBox

The HelpBox Component allows you to add Read-Only information to a GameObject. This can be used to explain confusing behavior or to instruct Designers how to make certain changes.



Lorem ipsum dolor sit amet consectetur adipisicing elit. Maxime mollitia,
molestiae quas vel sint commodi repudiandae consequuntur voluptatum
laborum
numquam blanditiis harum quisquam eius sed odit fugiat iusto fuga
praesentium
optio, eaque rerum! Provident similique accusantium nemo autem. Veritatis
obcaecati tenetur iure eius earum ut molestias architecto voluptate aliquam
nihil, eveniet aliquid culpa officia aut!

2.3.2 Inspector Attributes

The DebugTools Module includes a bunch of attributes to make your editor nicer:

- **[Button]:** Add this attribute to a Method to make it executable through a button in the inspector. This only works on parameterless methods. By default, the button will be named after the function. You can add a custom button name and height through the attribute's parameters.
- **[Foldout]:** Add this to a field to add it to a foldout in the inspector. You can specify the foldout group through a string parameter. Leaving it empty will add the field to the last declared group above it.
- **[ShowIf]:** Adding this to a field will only show it in the inspector if the set condition is true. The condition can be specified by passing in the name of a Boolean Field. It is recommended to use **"nameof(boolField)"** to avoid renaming issues. Through the second parameter the result can be inverted.

2.3.3 Gizmo Components

There are several Components you can use to quickly draw customized Gizmos:

- **BoxGizmo**
- **SphereGizmo**
- **RectGizmo**
- **CircleGizmo**
- **LineGizmo**

2.3.4 Gizmo Attributes

Usually, you don't want to change your gizmo parameters by hand but make them dependent on some variables in your code. To do this, you can use **Gizmo Attributes**, which hook into the Gizmo Components and overwrite their values. Simply add the attribute to a field and make sure your desired Gizmo Components are attached to the same GameObject as your script. You can even overwrite multiple Gizmos with the same script.

These are the available attributes:

- **[GizmoColor]:** The color. (Must be of type **Color**)
- **[GizmoOrigin]:** The position origin. (Must be of type **Transform**)
- **[GizmoOffset]:** The position offset from the origin. (Must be of type **Vector3**)
- **[GizmoRotationOffset]:** The rotation offset in Eulers. (Must be of type **Vector3**)
- **[GizmoSize]:** The size of a Box/Rect Gizmo. (Must be of type **Vector2** or **Vector3**, depending on whether the Gizmo is 2D or 3D).
- **[GizmoRadius]:** The radius of a Sphere/Circle Gizmo. (Must be of type **float**).
- **[GizmoSegments]:** The segment amount of a Circle Gizmo. (Must be of type **int**).
- **[GizmoTarget]:** The target/end of a Line Gizmo. (Must be of type **Transform**).
- **[GizmoTargetOffset]:** The position offset from the target. (Must be of type **Vector3**).

2.3.5 Custom Gizmos

If for whatever reason you want to draw these Gizmos without using the Components, you can use the ToolbeltGizmos class to invoke the same drawing methods that are used by the Gizmo Components.

2.4 Optimization

Currently, the Optimization Module consists of only a single class: The FrameSkipper.

Inherit from FrameSkipper to create a MonoBehaviour that can execute code in predefined frame-intervals. This can be used to spread expensive operations out over longer periods of time and thereby improve your framerate.

Once your class inherits from FrameSkipper, simply override the “DelayedUpdate” method. It is called whenever the set number of frames have passed. You can configure these values in the inspector (look at the provided Demo to gain more insight into the parameters). Especially note the “randomStartOffset” parameter. Using this, the first DelayedUpdate call is delayed by a random frame between 0 and your skip interval. This way, you can avoid a lot of DelayedUpdate’s being called in the same frame.

In case you still need the regular Update function or also the Awake function, make sure to call them with “override” and call **“base.Update()”** or **“base.Awake()”** respectively.

2.5 Scene Selector

To use the Scene Selector, navigate to:

“Window/StarBannerGames/Toolbelt/SceneSelector”

It should automatically load all Scenes in your Project and add them to the Selection List. Pressing Edit will allow you to change Display Names, Colors, Visibility and order of the Scene Buttons.

Known Bug: In certain edge cases I can’t reproduce, the Button Styling breaks. If the window suddenly looks weird, just click “Refresh”! If you figure out what circumstances cause this, please send me an email or reach out to me through one of the channels provided at the top of this manual. Thank you!

2.6 UI

2.6.1 ProgressBarUI

The ProgressBarUI class can be used to script UI Bars quickly. Just create a ProgressBarUI field and call **"SetValue()"** to update its state. It also supports a text element to show the current value as a string.

2.6.2 ResourceBarUI

ResourceBarUI is a component building on ProgressBarUI. It contains a complete implementation of a resource bar that can be used as a health bar, energy bar, etc.

It also includes an optional "Preview Bar", to make value jumps clearer to the player. If you want an easy example of how to use this, check out the provided demo.

You can very easily create a range of resource bar templates by right-clicking in the hierarchy window and selecting **"Toolbelt/UI/ResourceBar_YourChoice".**

2.6.3 ResourceCounterUI

The resource counter is a component that works practically the same as the resource bar, except that it is text-only. It also has some optional polishing effects in the form of a small bounce and color flash when its value changes. Check out the provided demo to see an example.

Again, same as with the resource bar, you can create one right from the hierarchy by right-clicking and selecting **"Toolbelt/UI/ResourceCounter".**

2.6.4 ScreenTransition

ScreenTransition is a very powerful class that lets you create various transition effects. All you need to do is call the provided static functions. It handles all the GameObjects internally, using **an Overlay Canvas with Sorting Layer 1000.** Keep this in mind if you want to render anything on top of the Transition. You can find the used Prefab in the Resource Folder, though it is very easy to break things if you decide to alter it.

Transitions can go from the game to a solid screen color, from the color to the game and from the previous color to a new one. Checkout the provided demo to get a better idea of this.

Here is a list of all available transitions:

- **ShowBlackscreen:** Instantly cuts to the specified screen color.
- **HideBlackscreen:** Instantly cuts to the game.
- **FadeToColor:** Fades to the specified screen color.
- **FadeToGame:** Fades to the game.
- **CutoutToColor:** Makes color flood into the screen with a specified cutout shape in the middle (defaults to a circle).
- **CutoutToGame:** Reverses the effect of **CutoutToColor** and ends back in the game.
- **WipeToColor:** Wipes the specified color on top of the game in the specified direction
- **WipeToGame:** Wipes the current color off the game in the specified direction

SetCutoutShape: This function lets you configure a sprite to use as the cutout shape, which will otherwise default to a circle. Only the alpha value is used to determine the cutout, so make sure the sprite has proper transparency. I also recommend keeping the aspect ratio at 1:1.

SetEdgeSprites: This function lets you configure what sprites to use as the edges of the Screen Wipe Effect. You can use an Enum of presets for this, as there are already a bunch of premade edges available. If you do want to create your own, you will need a **Leading** as well as a **Trailing** Sprite. The leading sprite is for the edge that ENTERS the screen. The trailing sprite for the edge that LEAVES the screen. These sprites should be solid white on a transparent background. The sprites are rotated internally, so you only need to create them facing **upwards**. Check out the resources folder to see how the presets are made. Keep the shapes simple as the edges will squash/stretch depending on whether the transition is vertical or horizontal. I'm sorry if you don't like this approach. I prefer simple shapes for the transitions and found this to be the best way to do it without having to create separate sprites for every direction and having to think about different screen resolutions.