Introduction

The SharpUI asset is developed with clean code in mind and is also unit tested. It should be noted that there are two library dependencies for this project, **UniRx** and **NSubstitute**. UniRx is unitys reactive extension library that supports most of the reactive features. NSubstitute is used for mocking in unit tests.

UniRx: https://github.com/neuecc/UniRx

NSubstitute: https://nsubstitute.github.io

UniRx Required (download mandatory)

UniRx is required by this project and is mandatory for this asset to work. You need to download it from the unity asset store and it is completely free of charge.

MVP (Model View Presenter)

This project is built with MVP pattern throughout the demo scenes and the base MVP setup scripts are part of the package. For the sake of Unity and because we thought it would make more sense, we have decided to call the View for Component, so what we basically have is MCP (Model Component Presenter).

Unit Tests

As this is the core UI library for our main game project we have decided to write unit tests for all the core features and also the demo scenes. The core components have **100%** code coverage with **700+** unit tests. Things that are not included in tests are prototype and view(component) scenes as those deserve manual UI testing.

Package content

Elements and components

- Model View Presenter base setup classes
- Advanced skill tree system with modular building elements
- Action Bar buttons and utilities
- Lists and Arrow Lists
- Selection strategies for lists
- Adapters for lists and Arrow Lists
- Buttons (Rectangular, Circular, Icon, Tab, Skill Tree)

- Decorators for button sprites, colors and texts (color, scale, offset, visibility, extendable)
- Dialogs with predefined buttons
- Drop down
- Input Text Fields
- Loading Bar
- Modal Views
- Notifications
- Progress Bars, Skill Bars and ResourceBars (customizable)
- Sliders
- Toggle buttons (checkmark / square)
- Button selection groups
- Tooltips with custom content

Utilities

- Canvas Group animation helper
- Keyboard input listener (works with all buttons and can also be standalone)
- Scene loading, unloading, additive with on complete callback
- Managed UI update to control the refresh rate of UI components
- Cooldown utils with time formatting and time consuming
- Collision detection for UI elements
- Password encryption tools
- Reactive delay observer

Scenes & Prefabs

All of the components provided in this package are part of the demo scenes. Some of the components have a dedicated prototype scene and prefabs. The root folder of all scenes and prefabs can be found at

Scenes: */Assets/SharpUI/Views*

Prefabs: */Assets/SharpUI/Prefabs*

Prototype and demo scenes should provide enough code to get you started using the components in your own project.

Buttons

Prefabs:

- **ActionBarButton.prefab**
- **ArrowButton.prefab**
- **IconButton.prefab**
- **LinkButton.prefab**
- **RectButton.prefab**
- **RoundButton.prefab**
- **TabButton.prefab**

Demo/Prefab Scenes:

- **Tabs.unity**
- **Buttons.unity** (prefab) and most off the other scenes contain buttons

Dialog

Prefabs:

- **DialogDefault.prefab**

Demo/Prefab Scenes:

- **Login.unity**
- **Dialogs.unity**

Input

Prefabs:

- **CheckBoxCheckmark.prefab**
- **CheckBoxSquare.prefab**
- **Dropdown.prefab**
- **InputField.prefab**
- **SimpleSlider.prefab**

Demo/Prefab Scenes:

- **Buttons.unity**
- **Settings.unity**

List

Prefabs:

- **ArrowList.prefab**
- **SharpList.prefab**
- **ListItemCustom.prefab**
- **ListItemDescription.prefab**
- **ListItemImage.prefab**
- **ListItemText.prefab**

Demo/Prefab Scenes:

- **SharpList.unity**
- **CharacterSelection.unity**

Progress

Prefabs:

- **LoadingBar.prefab**
- **ResourceBar.prefab**
- **SkillBar.prefab**

Demo/Prefab Scenes:

- **LoadingScene.unity**
- **GamePlayground.unity**

Skill Tree

Prefabs:

- **SkillTreeAbilityAmount.prefab**
- **SkillTreeAmountLimit.prefab**
- **SkillTreeArrowLine.prefab**
- **SkillTreeButton.prefab**
- **SkillTreeNode.prefab**
- **SkillTreeProgressLine.prefab**

Demo/Prefab Scenes:

- **SkillTree.unity**
- **GamePlayground.unity**

Modal View

Prefabs:

- **ModelView.prefab**

Demo/Prefab Scenes:

- **GamePlayground.unity**
- **ModalView.unity**

Notification

Prefabs:

- **Notification.prefab**

Demo/Prefab Scenes:

- **GamePlayground.unity**

Tooltip

Prefabs:

- **Tooltip.prefab**

Demo/Prefab Scenes:

- **Tooltip.unity**
- **SkillTree.unity**
- **GamePlayground.unity**

How To Use

Buttons

Most of the buttons have prefabs so check those as described in the **Scenes & Prefabs** section. All buttons inherit from BaseElement and you can change the buttons initial state in the editor. There are 4 states: enabled, clickable, selectable and selected. You can subscribe to button events with BaseElement.GetEventListener(). There are following observables:

- **ObserveOnPressed();**
- **ObserveOnReleased();**
- **ObserveOnClicked();**
- **ObserveOnLeftClicked();**
- **ObserveOnRightClicked();**
- **ObserveOnMiddleClicked();**
- **ObserveOnEnabled();**
- **ObserveOnDisabled();**
- **ObserveOnEntered();**
- **ObserveOnExited();**
- **ObserveOnSelected();**
- **ObserveOnDeselected();**

To read the button state use BaseEvent.GetState() instead of using the SerializedField variables as those are primarily used for initialization.

CircleButton has dedicated collision detection and is not based on background's alpha, which means that you can have any kind of background (event transparent button) and the click will still work.

Decorators

Decorators are designed to work with any BaseElement button and are quite powerful once you understand how they work. The main idea behind decorators is that they work together with button events like, enable/disable, press/release, hover/exit and select/deselect. You can add decorators to Images and TextMeshPro texts to modify the color(and sprite for image) of those.

There are other decorators like ScaleDecorator, OffsetDecorator and VisibilityDecorator. You can easily create your own decorator by extending the BaseDecorator<T> generic class. The implementation should be straight forward and you can see how the other decorators have been implemented to get the idea.

BaseDecorator<T> is also MonoBehaviour so you can add the decorators directly in the editor. Most of the buttons in the scenes have decorators attached to either TextMeshPro elements or the Image sprites, and you can check most of the scenes to see how that is done.

Skill Trees

You can build your own custom skill tree without the need to do any coding. Here we will describe in detail how you can create and design your skill tree.

Future

We have big plans for **SharpUI** as this is our main UI for the upcoming game that is currently in development. Here is a future list of some of the features:

- Editor scripts for real time updates while editing the UI elements
- Inventory system with drag and drop
- Minimap window

These are just some of the features that are in development at the moment.

Support & Updates

You can always contact us if you find any bugs or challenges while using this library. Keep in mind that this documentation will be updated and the most recent version can be found at http://www.boki.dk/sharpui

Contact

Product Support: support@boki.dk