Mobile Controller Toolbox

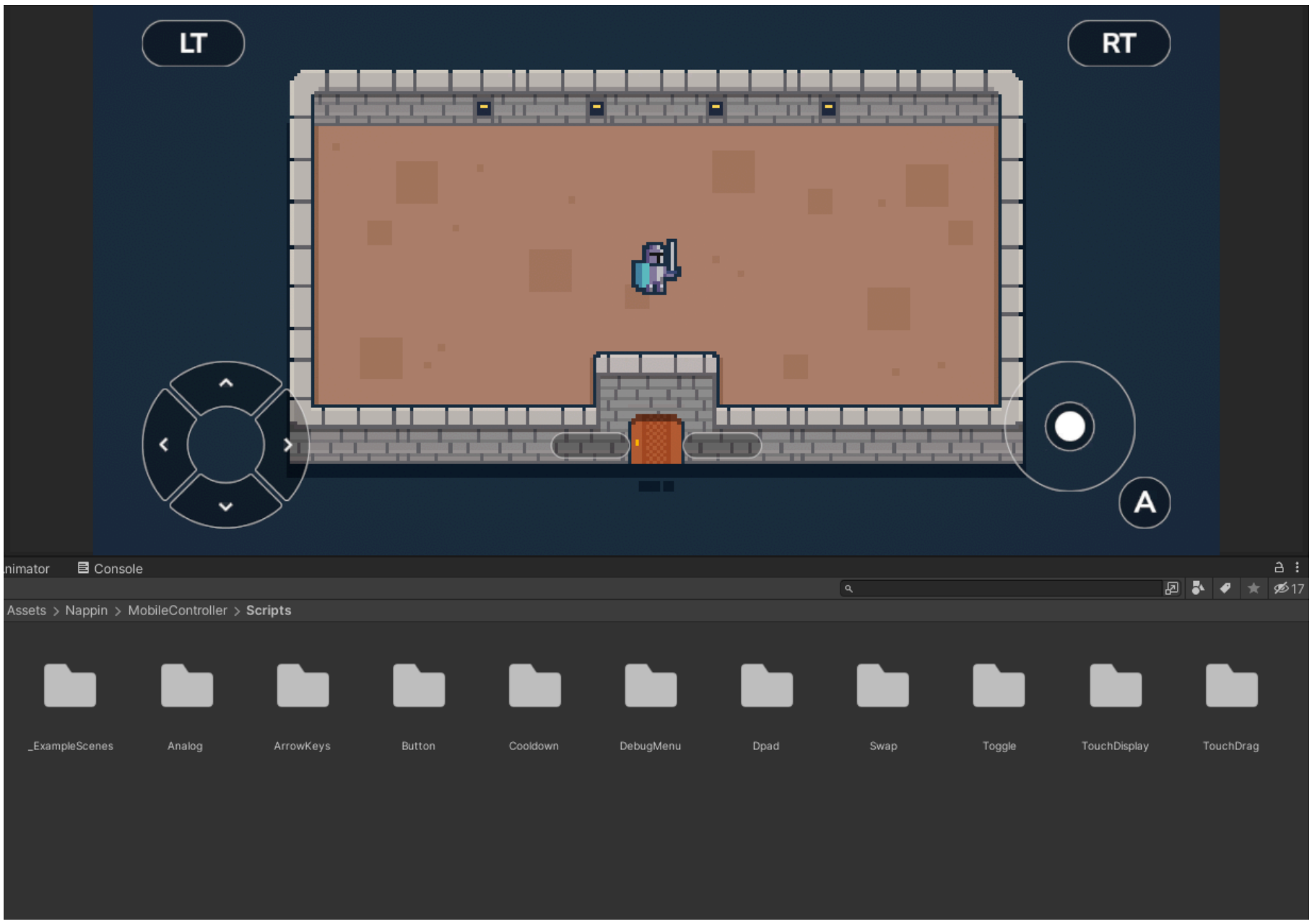
*Support email: **nappin.1bit@gmail.com***

Table of Contents

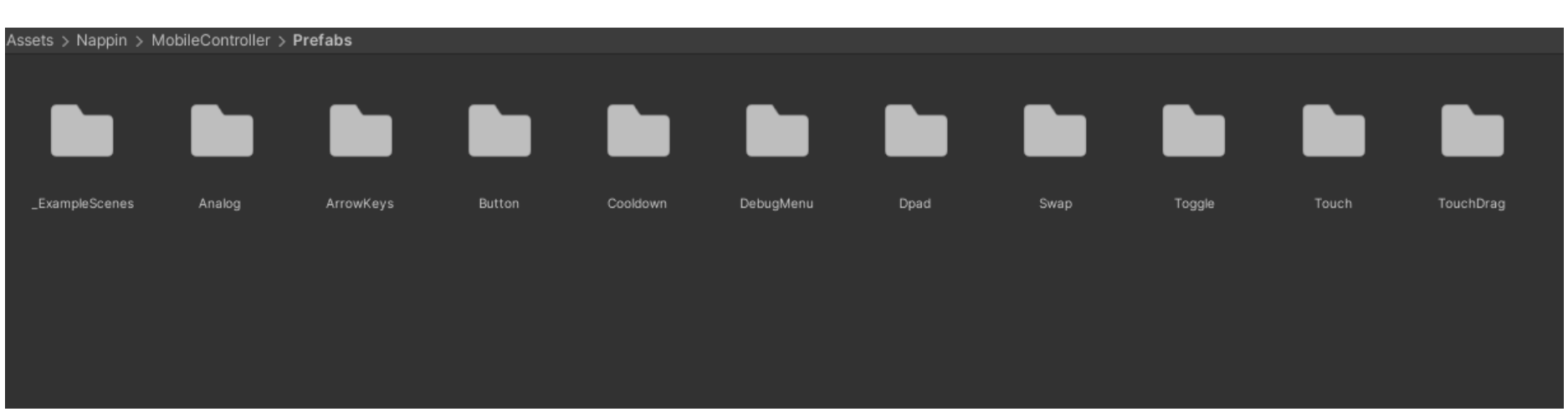
Asset Content

Mobile Controller Toolbox is a collection of highly customizable, commonly used Mobile *Input Types*. The Mobile Controller Toolbox core content can be found in the *Prefabs* folder and in the *Scripts* folder. The following documentation will touch on the core content of the Mobile Controller Toolbox package and give an exhaustive overview of the various *Input Types* available. The package additionally includes 2 sample scenes to show practical use of the various *Input Types*.

In the *Scripts* folder you can find **Analog, ArrowKeys, Button, Cooldown, DebugMenu, Dpad, Swap, Toggle, TouchDisplay and TouchDrag** folders which contain the relevant scripts. Additionally you can find the **_ExampleScenes** folder, which contains scripts used in the sample scenes that are not core to the package.

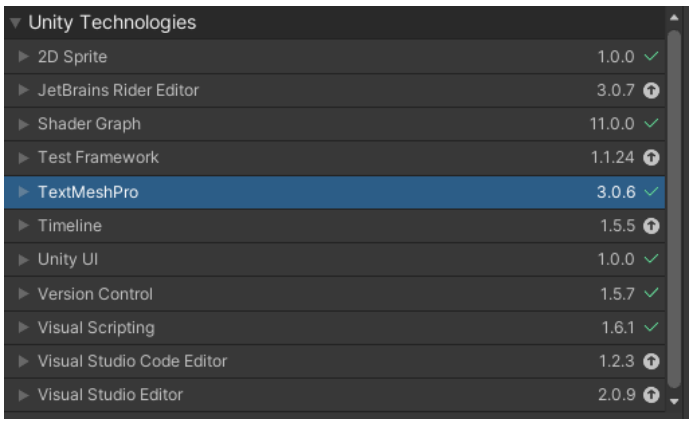


In the *Prefab* the content is organized following the same foldering.



External Packages

Mobile Controller Toolbox doesn't require any external package. TextMeshPro is used in Mobile Controller Toolbox to render texts but it's not required, the default Unity Text Component can be used instead (or any other text renderer). Additionally both the sample scenes contain a PostProcessing layer, the package is only visual and not required.

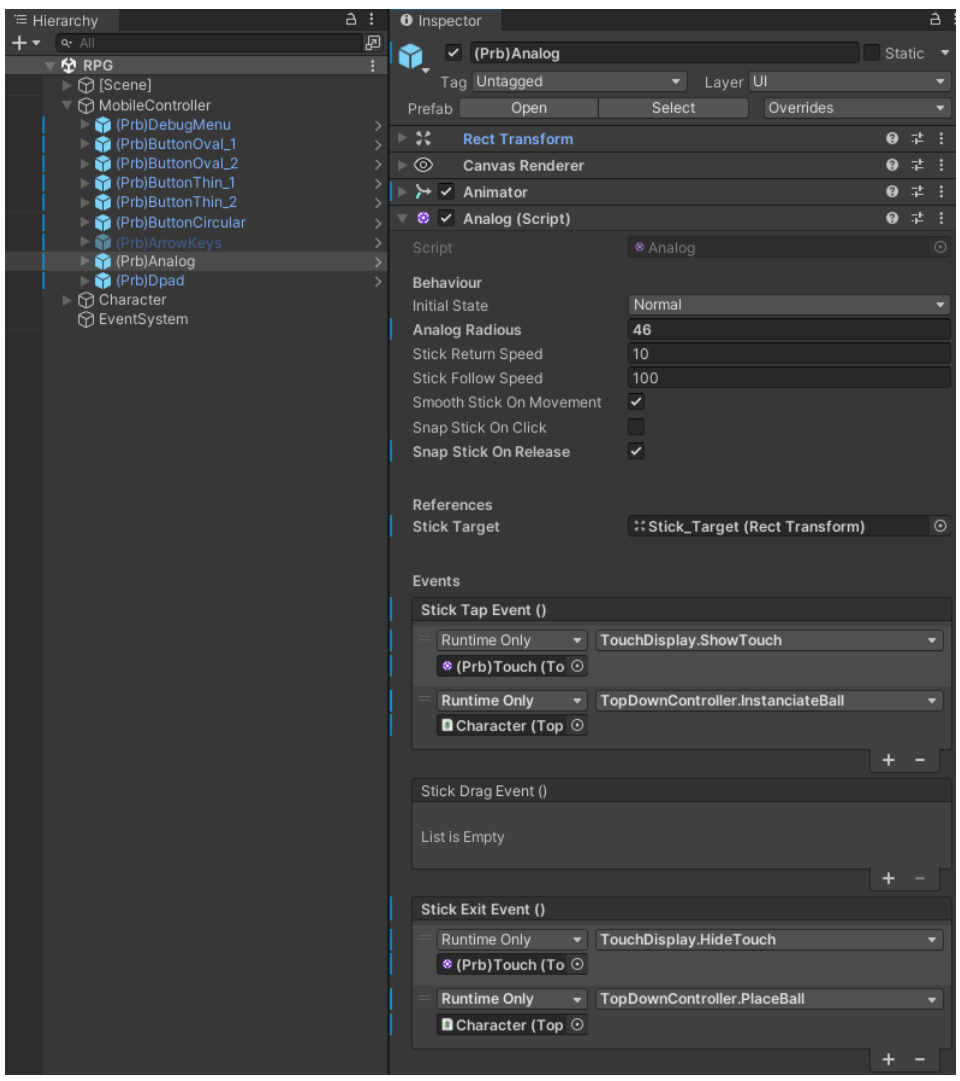


P.S. If you intend to use TextMeshPro, there is no need to install it before downloading Mobile Controller Toolbox. When the Mobile Controller Toolbox package is imported the install prompt for TextMeshPro will be displayed automatically.

Input Types Structure

All the *Input Types* have the following structure:

- ***Animator:*** the visual of each *Input Type* is controlled via Animator Component. It's quite easy to edit the animation used for the various states (Normal, Tap, Disable, etc..) or to create custom Animators with different animations.
- ***Input Components:*** every *Input Type* relies on a different component. In every prefab this component can always be found inside the *Input Type* parent.



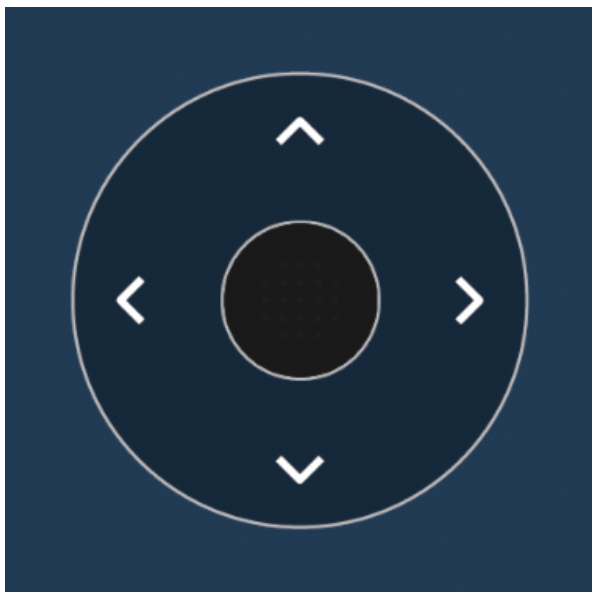
- ***Input Hierarchy:*** all the *Input Types* have a consistent hierarchy and follow a simple rule: the input detection and the input visual are separated. The input detection is handled by a transparent image, while the input visual is managed separately. With this approach managing customisation and setting dead zones is much easier.
- ***Special Children:*** some *Input Types* (like Dpad, ArrowKeys, Swap) contain special children. More detail in the following sections.

Input Types

The following is a list of all the *Input Types* available in the Mobile Controller Toolbox package. New *Input Types* can be created tweaking the existing one. Additionally, as shown in the sample scenes, the same *Input Type* can be used for different purposes, with different visuals in different ways.

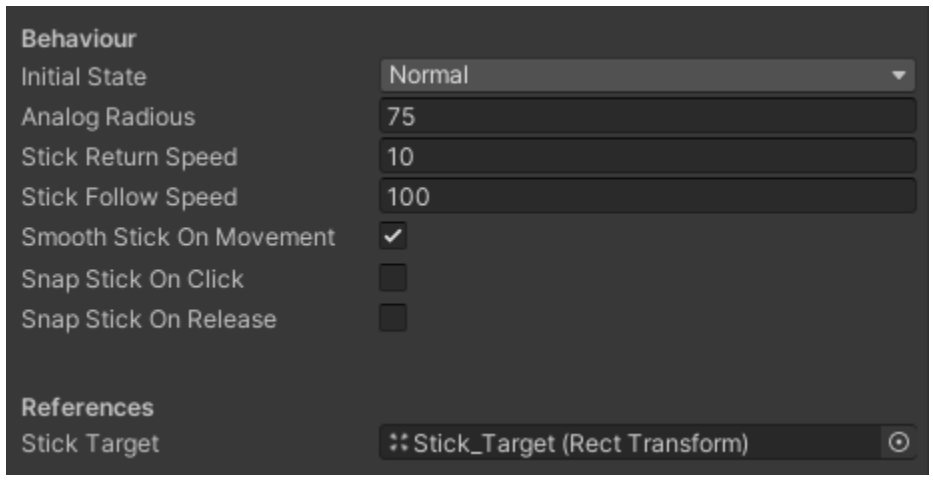
Analog

The Analog *Input Type* uses the position of a draggable shape (Analog Stick) relative to a background shape to generate a Vector2.



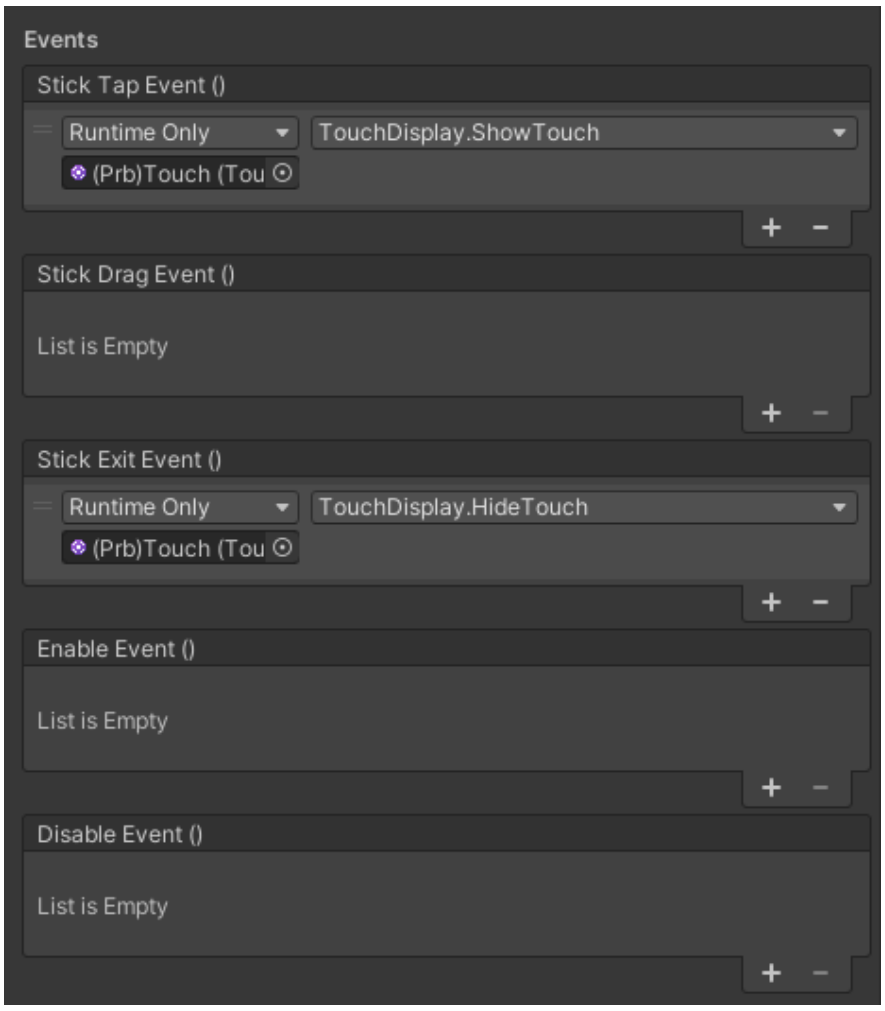
Analog Component

The following is an overview of what is available in the Analog Component.



Its variables are:

- ***Initial State***: this property can be set as Normal or Disable. It determines the Analog starting state.
- ***Analog Radious:*** determines the circumference in which the Analog Stick can move.
- ***Stick Return Speed***: the speed in which the Analog Stick returns to its (0,0) position after the player lets go.
- ***Stick Follow Speed:*** the speed used by the Analog Stick to reach the touch position.
- ***Smooth Stick On Movement***: determines if smoothness is applied to the Analog Stick motion.
- ***Snap Stick On Click***: determines if the Analog Stick follows the touch position from its center or from where the touch occurred.
- ***Snap Stick On Release***: determines if the Analog Stick returns to its (0,0) position smoothly after the player lets go.
- ***Stick Target***: reference to the Analog Stick GameObject.



Its events are:

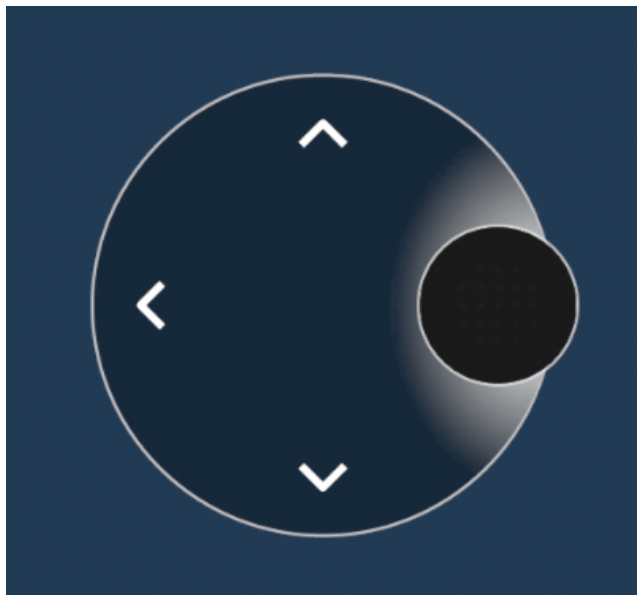
- ***Stick Tap Event***: called when the Analog Stick is touched (if Analog is not in the Disable state).
- ***Stick Drag Event***: called when the position of the Analog Stick changes.
- ***Stick Exit Event***: called when the player lets go of the Analog Stick.
- ***Enable Event***: called when the Analog is enabled via code.
- ***Disable Event***: called when the Analog is disabled via code.

Its relevant methods are:

- ***GetStickPosition()***: returns the position of the Analog Stick in a Vector2.
- ***GetAnalogState():*** returns the *AnalogState* of the Analog.
- ***Enable***: enables the Analog.
- ***Disable***: disables the Analog.

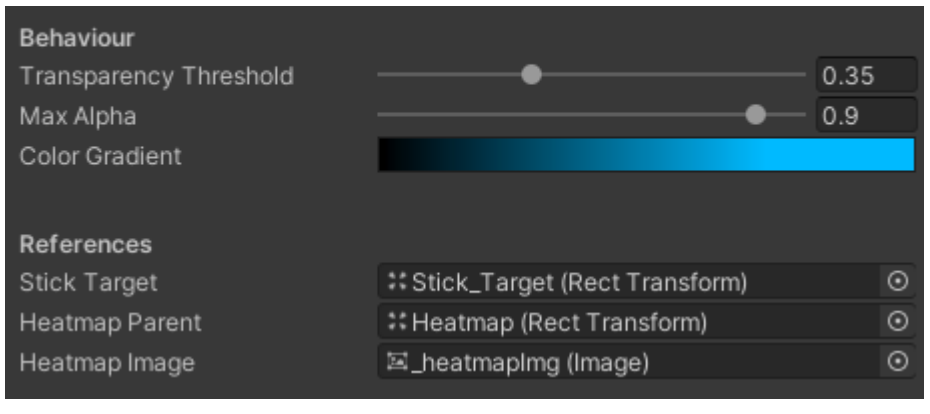
Heatmap

The Heatmap is an additional component that can be added only to GameObject containing the Analog Component. Heatmap allows the player to visualize the magnitude of the Analog Stick Vector2.



Heatmap Component

The following is an overview of what is available in the Heatmap Component.



Its variables are:

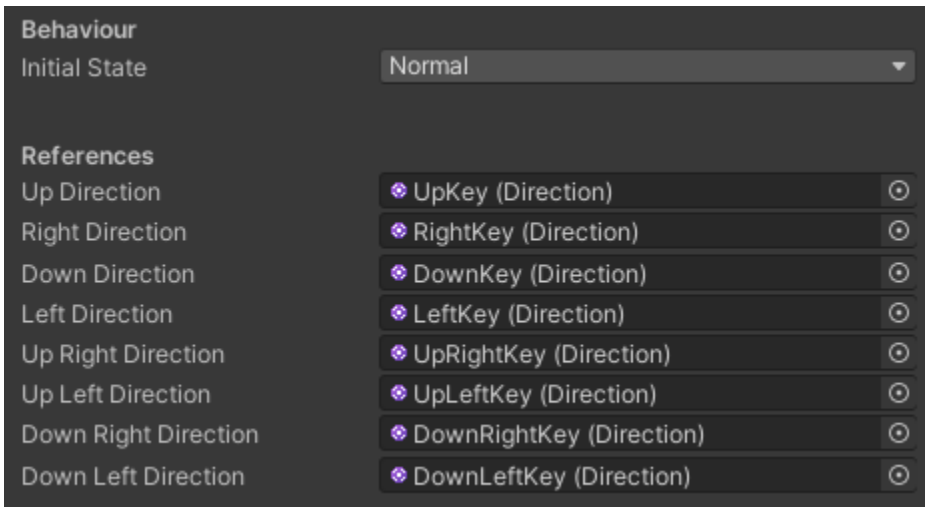
- ***Transparency Threshold***: which is the value in which the Heatmap Image will stop being fully transparent.
- ***Max Alpha:*** the maximum value of transparency that the Heatmap Image can reach.
- ***Color Gradient***: the color of the Heatmap Image from minimum to maximum magnitude.
- ***Stick Target***: reference to the Analog Stick GameObject.
- ***Heatmap Parent***: the parent of the Heatmap Image. The parent should alway be centered in the Analog.
- ***Heatmap Image***: the Image Component used to display the Heatmap magnitude.

Dpad

The Dpad *Input Type* uses 8 interactable areas (4 cardinal and 4 diagonal) to generate a Vector2Int based on what the player is touching. The Dpad *Input Type* contains in its hierarchy 8 children elements with the Direction Component.

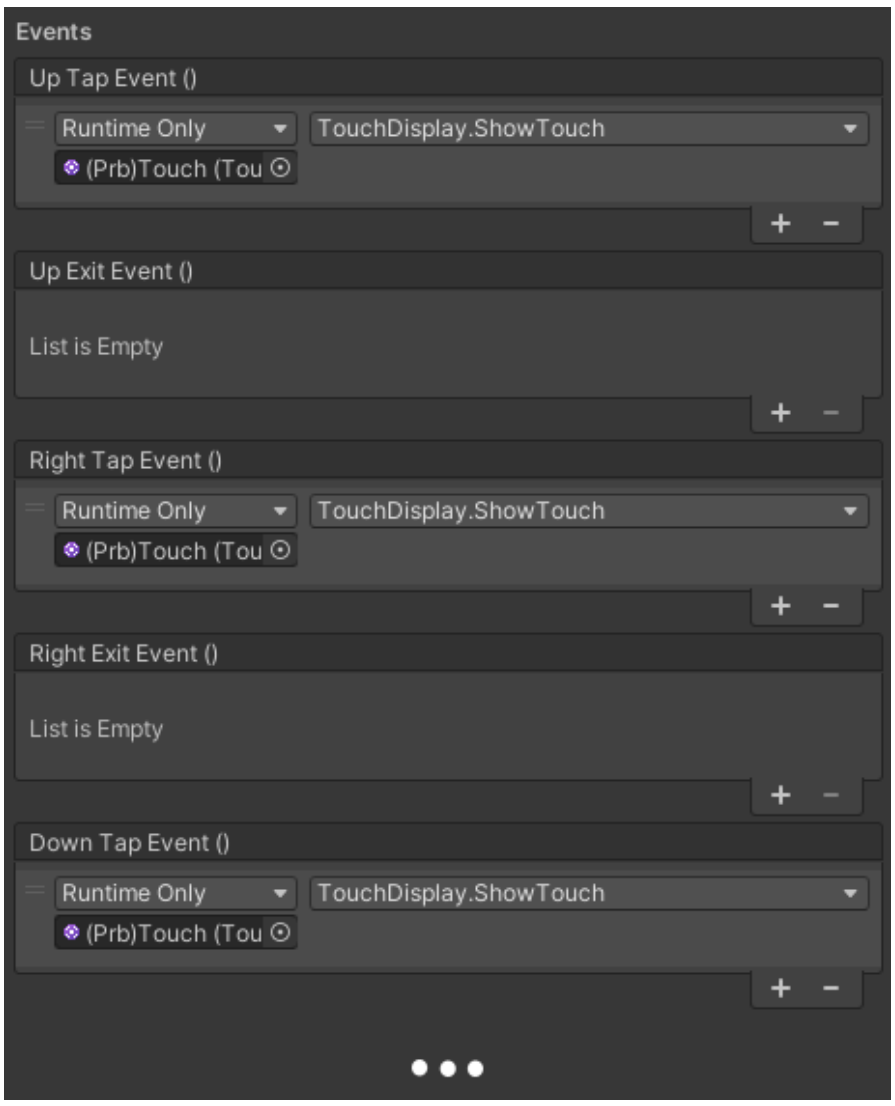
Dpad Component

The following is an overview of what is available in the Dpad Component.



Its variables are:

- ***Initial State***: this property can be set as Normal or Disable. It determines the Dpad starting state.
- ***Up Direction:*** reference to the Direction Component that is used to detect Up.
- ***Right Direction:*** reference to the Direction Component that is used to detect Right.
- ***Down Direction:*** reference to the Direction Component that is used to detect Down.
- ***Left Direction:*** reference to the Direction Component that is used to detect Left.
- ***Up Right Direction:*** reference to the Direction Component that is used to detect Up - Right.
- ***Up Left Direction:*** reference to the Direction Component that is used to detect Up - Left.
- ***Down Right Direction:*** reference to the Direction Component that is used to detect Down - Right.
- ***Down Left Direction:*** reference to the Direction Component that is used to detect Down - Left.



Its events are:

- ***Up Tap Event***: called when the Up Direction is considered pressed (if Dpad is not in the Disable state).
- ***Up Exit Event***: called when the Up Direction is considered no longer pressed.
- ***Right Tap Event***: called when the Right Direction is considered pressed (if Dpad is not in the Disable state).
- ***Right Exit Event***: called when the Right Direction is considered no longer pressed.
- ***Down Tap Event***: called when the Down Direction is considered pressed (if Dpad is not in the Disable state).
- ***Down Exit Event***: called when Down Direction is considered no longer pressed.
- ***Left Tap Event***: called when the Left Direction is considered pressed (if Dpad is not in the Disable state).
- ***Left Exit Event***: called when Left Direction is considered no longer pressed.
- ***Enable Event***: called when the Dpad is enabled via code.
- ***Disable Event***: called when the Dpad is disabled via code.

Its relevant methods are:

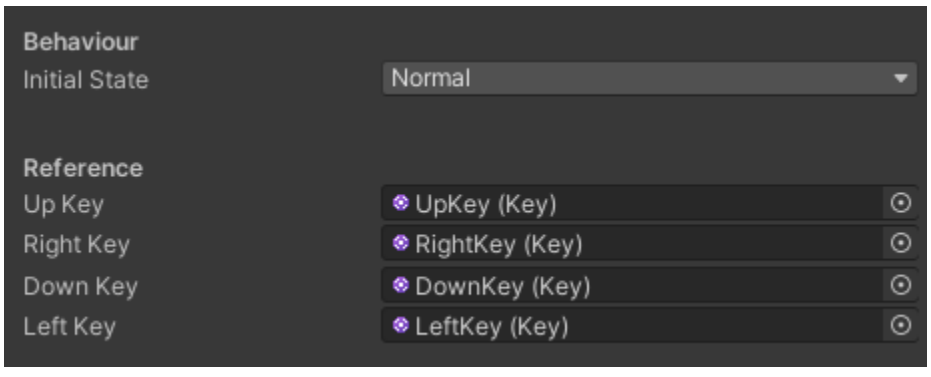
- ***GetDpadDirection()***: returns the Vector2Int direction of the Dpad.
- ***GetDpadState():*** returns the *DpadState* of the Dpad.
- ***Enable***: enables the Dpad.
- ***Disable***: disables the Dpad.

Arrow Keys

The Arrow Keys *Input Type* uses 4 interactable areas (all cardinal) to generate a Vector2Int based on what the player is touching. The Arrow Keys *Input Type* contains in its hierarchy 4 children elements with the Key Component.

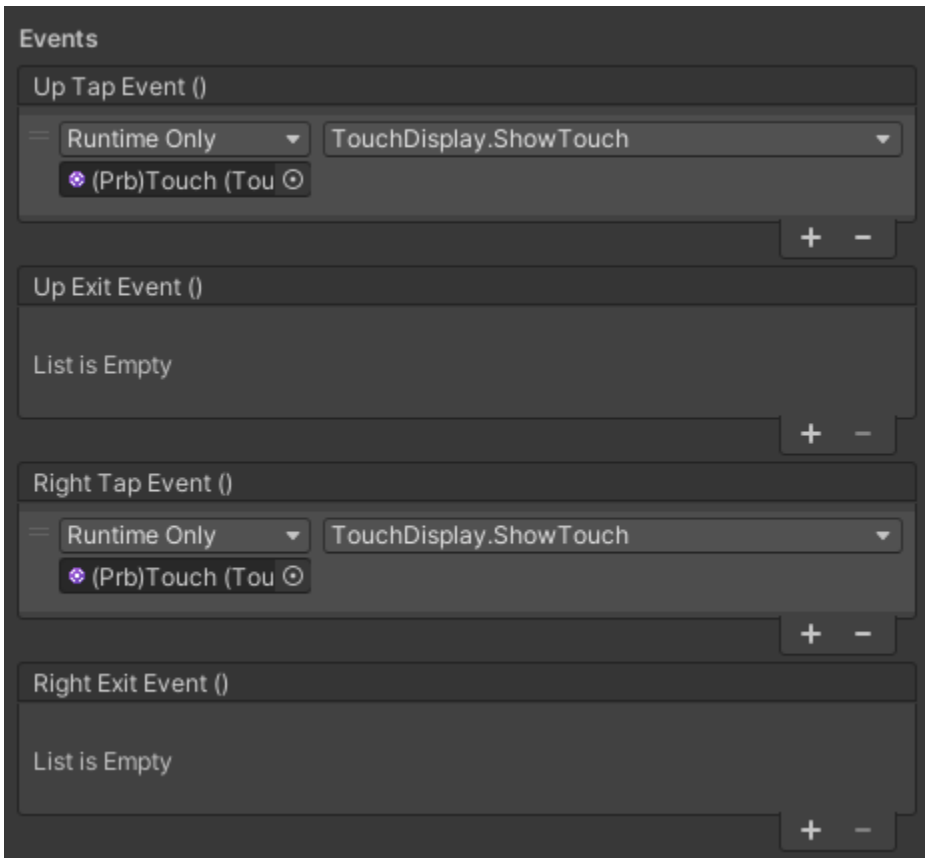
Arrow Keys Component

The following is an overview of what is available in the Arrow Keys Component.



Its variables are:

- ***Initial State***: this property can be set as Normal or Disable. It determines the Arrow Keys starting state.
- ***Up Key:*** reference to the Key Component that is used to detect Up.
- ***Right Key:*** reference to the Key Component that is used to detect Right.
- ***Down Key:*** reference to the Key Component that is used to detect Down.
- ***Left Key:*** reference to the Key Component that is used to detect Left.



Its events are:

- ***Up Tap Event***: called when the Up Key is considered pressed (if Arrow Keys is not in a Disable state).
- ***Up Exit Event***: called when the Up Key is considered no longer pressed.
- ***Right Tap Event***: called when the Right Key is considered pressed (if Arrow Keys is not in a Disable state).
- ***Right Exit Event***: called when the Right Key is considered no longer pressed.
- ***Down Tap Event***: called when the Down Key is considered pressed (if Arrow Keys is not in a Disable state).
- ***Down Exit Event***: called when Down Key is considered no longer pressed.
- ***Left Tap Event***: called when the Left Key is considered pressed (if Arrow Keys is not in a Disable state).
- ***Left Exit Event***: called when Left Key is considered no longer pressed.
- ***Enable Event***: called when the ArrowKeys is enabled via code.
- ***Disable Event***: called when the ArrowKeys is disabled via code.

Its relevant methods are:

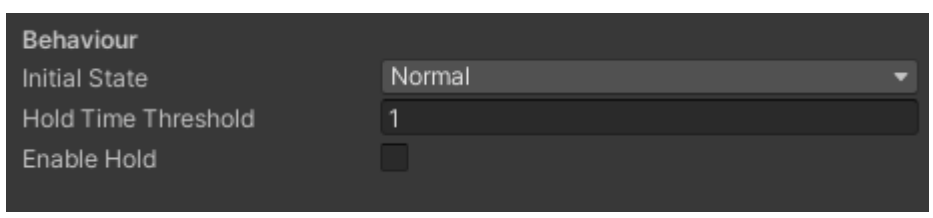
- ***GetArrowKeysDirection()***: returns the Vector2Int direction of the Arrow Keys.
- ***GetArrowKeysState():*** returns the *ArrowKeysState* of the Arrow Keys.
- ***Enable***: enables the Arrow Keys.
- ***Disable***: disables the Arrow Keys.

Button

The Button *Input Type* uses an interactable area to determine if the player performed a tap or an hold.

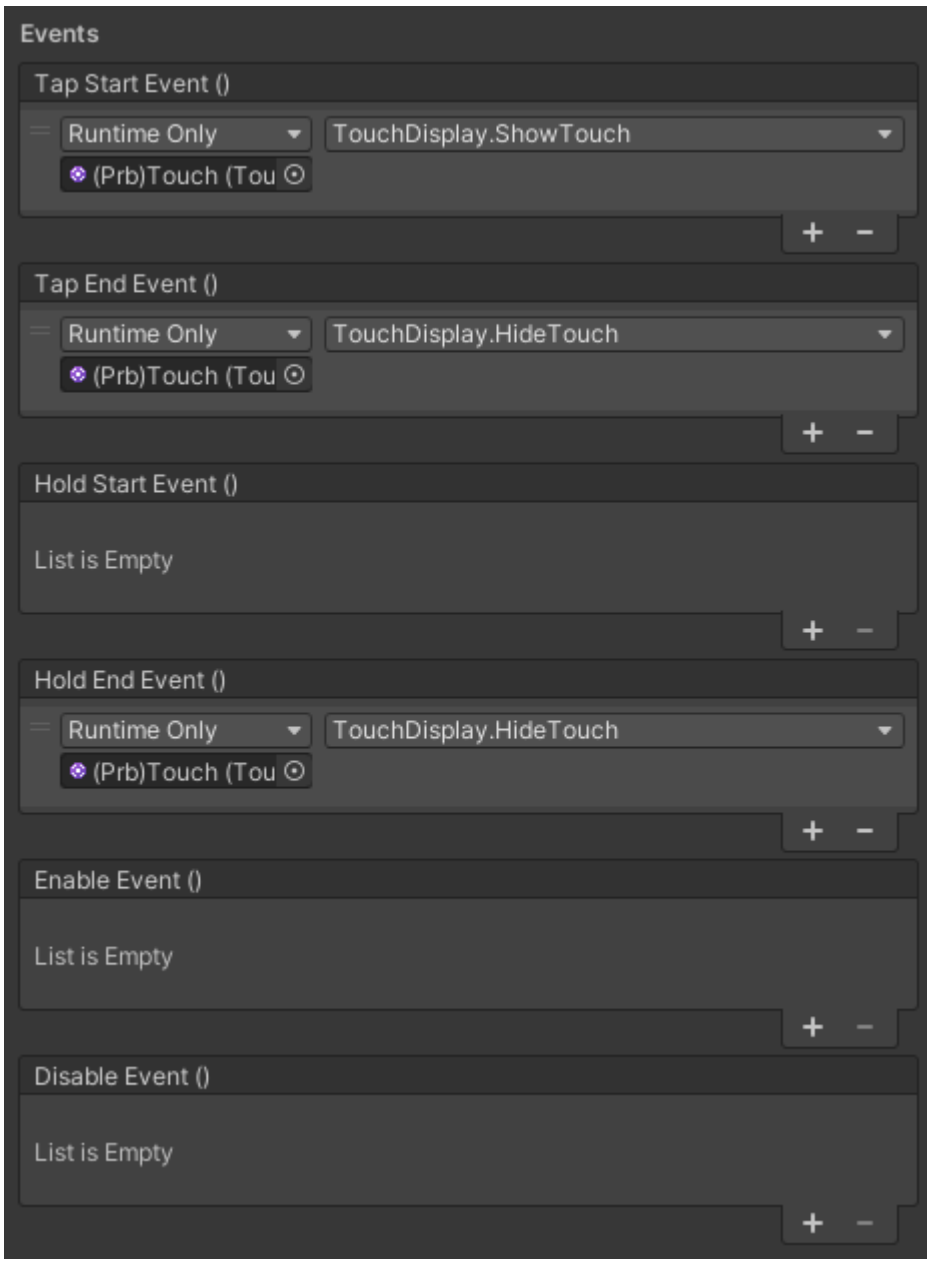
Button Component

The following is an overview of what is available in the Button Component.



Its variables are:

- ***Initial State***: this property can be set as Normal or Disable (if Button is not in a Disable state).
- ***Hold Time Threshold:*** the time required to trigger an Hold.
- ***Enable Hold:*** determines if, in addition to a Tap, an Hold can be performed.



Its events are:

- ***Tap Start Event***: called when the player taps the Button.
- ***Tap End Event***: called when the player lets go of the Button.
- ***Hold Start Event***: called when the Hold starts.
- ***Hold End Event***: called when the Hold ends.
- ***Enable Event***: called when the Button is enabled via code.
- ***Disable Event***: called when the Button is disabled via code.

Its relevant methods are:

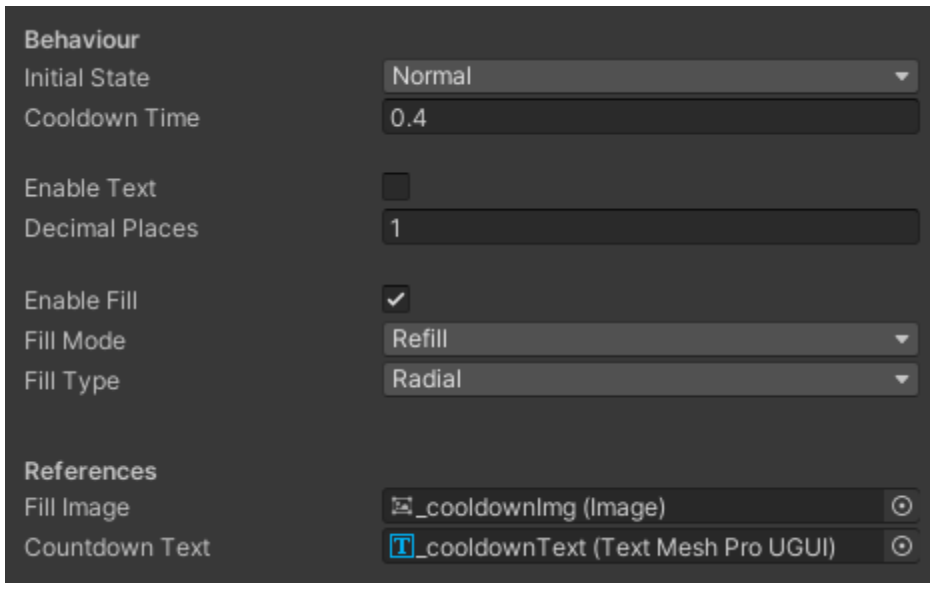
- ***GetButtonState():*** returns the *ButtonState* of the Button.
- ***Enable***: enables the Button.
- ***Disable***: disables the Button.

Cooldown

The Cooldown *Input Type* uses an interactable area to determine if the player performed a Tap. Once a tap is performed a countdown (with visual feedback) is started. When the countdown reaches 0, Cooldown can be pressed again.

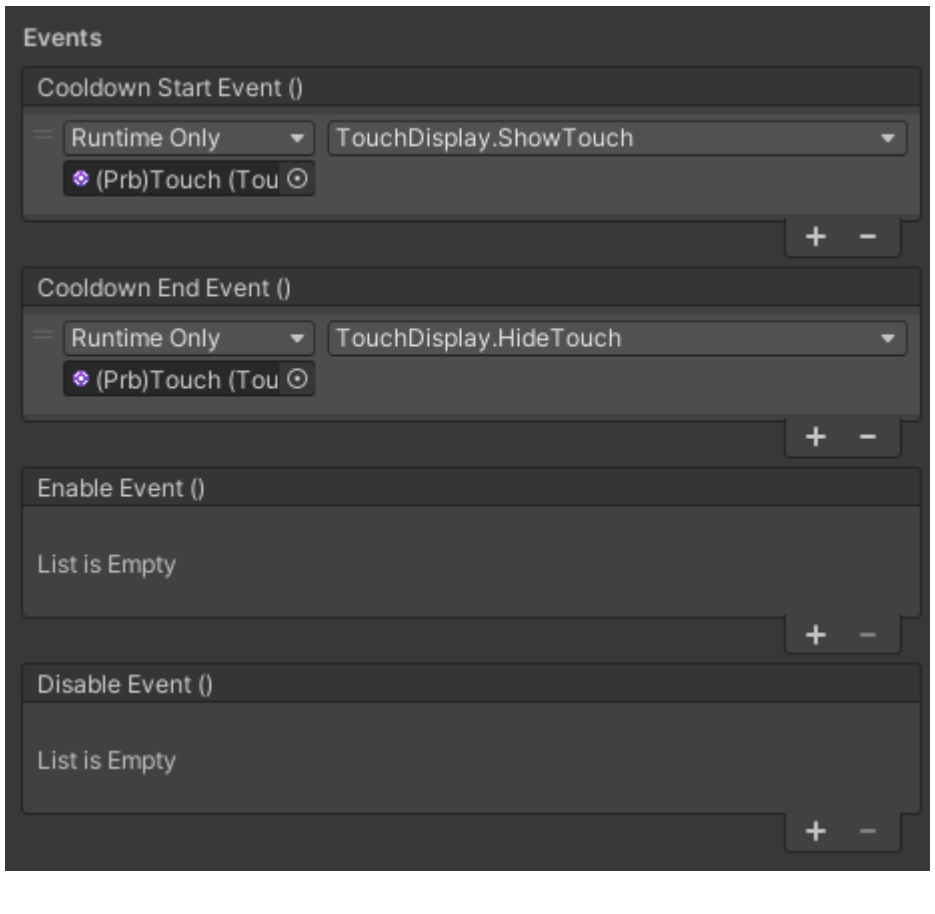
Cooldown Component

The following is an overview of what is available in the Cooldown Component.



Its variables are:

- ***Initial State***: this property can be set as Normal or Disable. It determines the Cooldown starting state.
- ***Cooldown Time:*** the duration of the countdown.
- ***Enable Text:*** if the boolean is true a text is displayed showing the countdown.
- ***Decimal Places:*** the number of decimal places displayed in the countdown text.
- ***Enable Fill:*** if the boolean is true a fill image is shown during the countdown.
- ***Fill Mode:*** if the property is set to *Refill* the fill image starts from 0 and reaches 1, if the property is set to *Empty* the fill image starts from 1 and reaches 0.
- ***Fill Type:*** property that manages the kind of image fill used (Radial, Horizontal, Vertical).
- ***Fill Image:*** the Image Component used for the fill.
- ***CountDown text:*** the Text Component used for the countdown.



Its events are:

- ***Cooldown Start Event***: called when the countdown starts.
- ***Cooldown End Event***: called when the countdown ends.
- ***Enable Event***: called when the Cooldown is enabled via code.
- ***Disable Event***: called when the Cooldown is disabled via code.

Its relevant methods are:

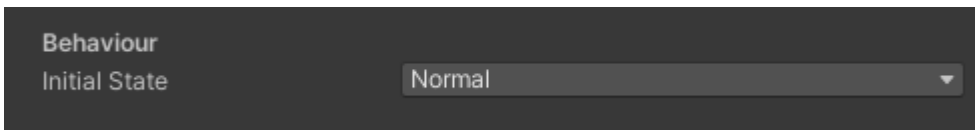
- ***GetCooldownState():*** returns the *CooldownState* of the Cooldown.
- ***GetCooldownTime():*** returns the countdown time.
- ***ClearCooldown():*** resets the countdown time to the initial value. The Cooldown is clickable again.
- ***Enable***: enables the Cooldown.
- ***Disable***: disables the Cooldown.

Toggle

The Toggle *Input Type* uses an interactable area to determine if the player performed a Tap. Once a Tap is performed the state of the Toggle is flipped (on from off or vice versa).

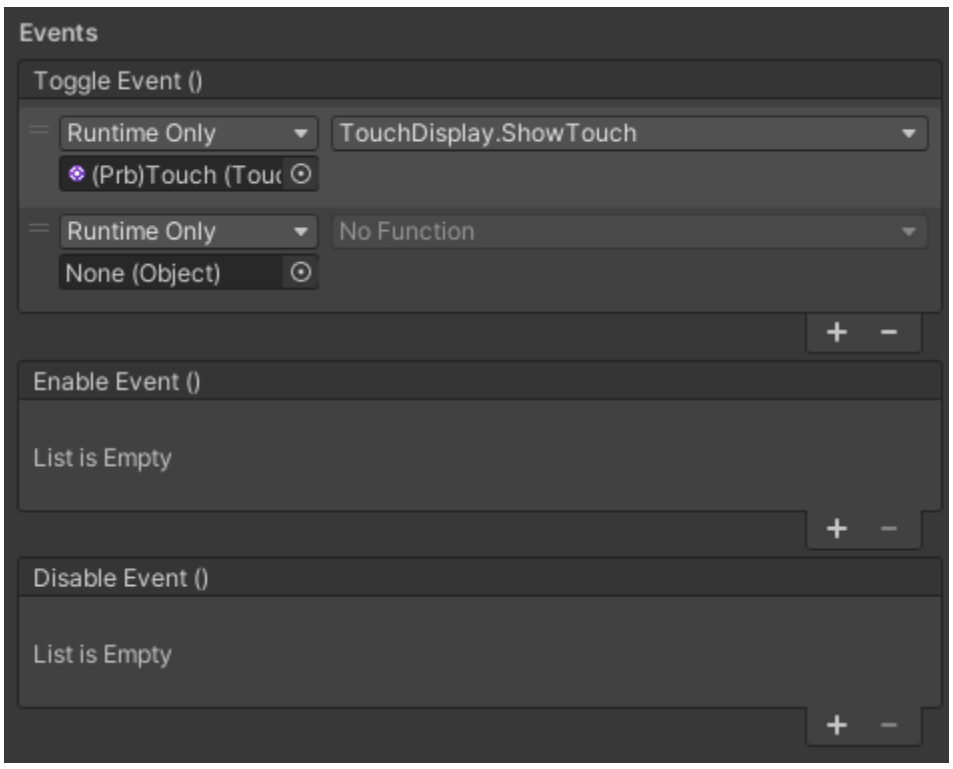
Toggle Component

The following is an overview of what is available in the Toggle Component.



Its variables are:

- ***Initial State***: this property can be set as Normal or Disable. It determines the Toggle initial state.



Its events are:

- ***Toggle Event***: called when a Toggle is performed.
- ***Enable Event***: called when the Toggle is enabled via code.
- ***Disable Event***: called when the Toggle is disabled via code.

Its relevant methods are:

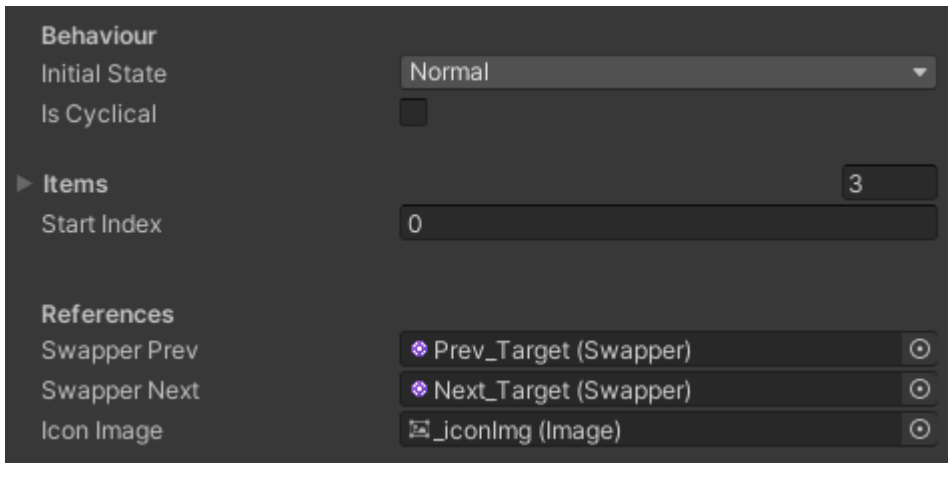
- ***GetToggleState():*** returns the *ToggleState* of the Toggle.
- ***Enable***: enables the Toggle.
- ***Disable***: disables the Toggle.

Swap

The Swap *Input Type* uses 2 interactable areas (containing the Swapper Component) to determine if the player pressed *Next* or *Previous*. Depending on the player input the current active “item” is swapped for the one before or after in a list (in the package is a list of Scriptable Objects but can be easily tweaked to fit your needs).

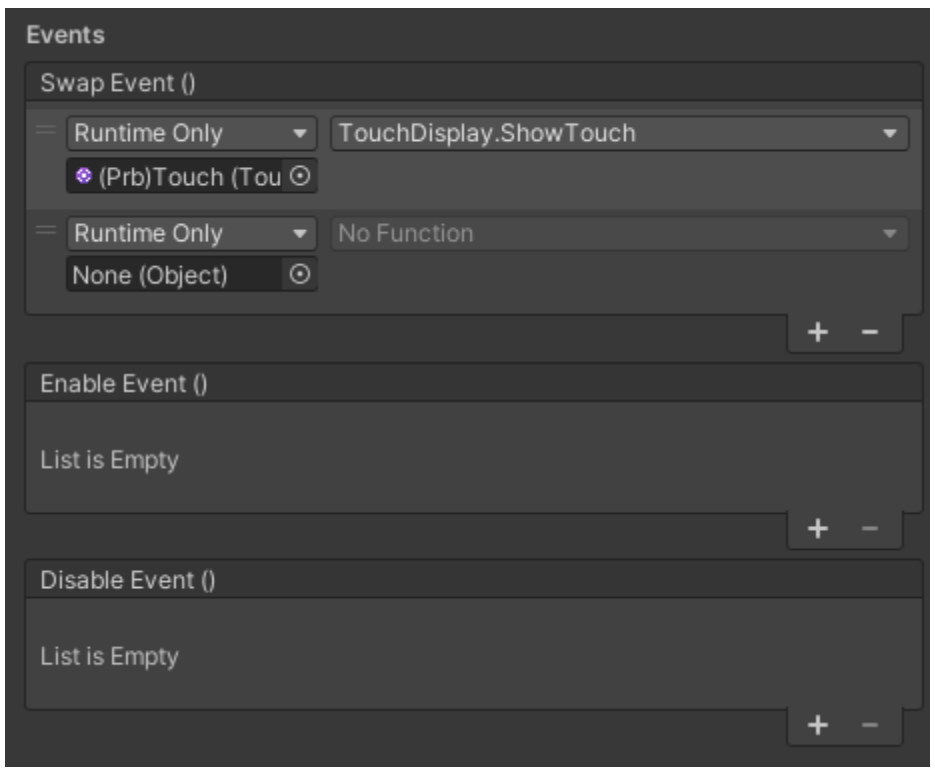
Swap Component

The following is an overview of what is available in the Swap Component.



Its variables are:

- ***Initial State***: this property can be set as Normal or Disable. It determines the Swap initial state.
- ***Is Cyclical***: this property determines if the list is loopable on both sides. If the boolean is set to true, the player can navigate indefinitely on the left and on the right.
- ***Items***: the content used inside the Swap.
- ***Start Index***: the index of the Item considered as first.
- ***Swapper Prev***: the Swapper used to select the previous Item.
- ***Swapper Next***: the Swapper used to select the next Item.
- ***Icon Image***: the Image Component used to display the current Item.



Its events are:

- ***Swap Event***: called when the current Item changes.
- ***Enable Event***: called when the Swap is enabled via code.
- ***Disable Event***: called when the Swap is disabled via code.

Its relevant methods are:

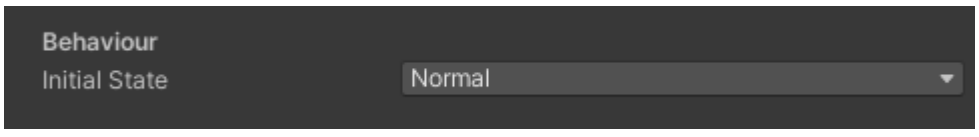
- ***GetCurrentItem():*** returns the current Item.
- ***GetCurrentIndex()***: returns the current Item index.
- ***GetSwapState()***: returns the *SwapState* of the Swap.
- ***Enable***: enables the Swap.
- ***Disable***: disables the Swap.

Touch Drag

The Touch Drag *Input Type* uses an interactable area to determine if the player touched it. Touch Drag uses the distance between when the touch is performed and when the finger is lifted to produce a Vector2 value (ideal for camera pans). Custom Touch Drag shapes can be realized simply using multiple *Targets*.

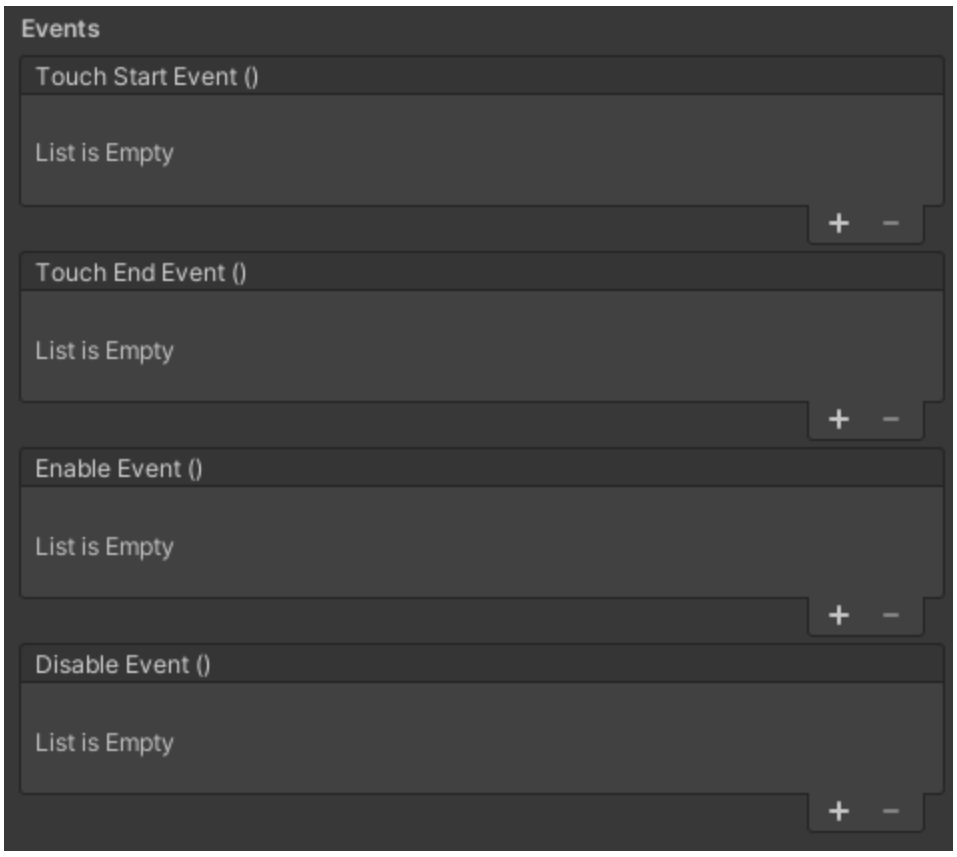
TouchDrag Component

The following is an overview of what is available in the Touch Drag Component.



Its variables are:

- ***Initial State***: this property can be set as Normal or Disable. It determines the Touch Drag initial state.



Its events are:

- ***Touch Start Event***: called when a touch is performed.
- ***Touch End Event***: called when the touch ends.
- ***Enable Event***: called when the TouchDrag is enabled via code.
- ***Disable Event***: called when the Touch Drag is disabled via code.

Its relevant methods are:

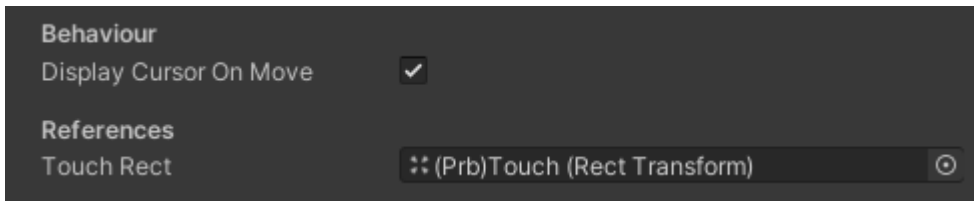
- ***GetTouchDragMovement():*** returns the Vector2 of the Touch Drag.
- ***GetTouchDragState():*** returns the *TouchDragState* of the Touch Drag.
- ***Enable***: enables the Touch Drag.
- ***Disable***: disables the Touch Drag.

Touch

The Package contains a support *Prefab* called touch used to display the current “touch” input on PC. Touch only supports one input at the time (which is not problematic for Editor Debug).

Touch Component

The following is an overview of what is available in the Touch Component.

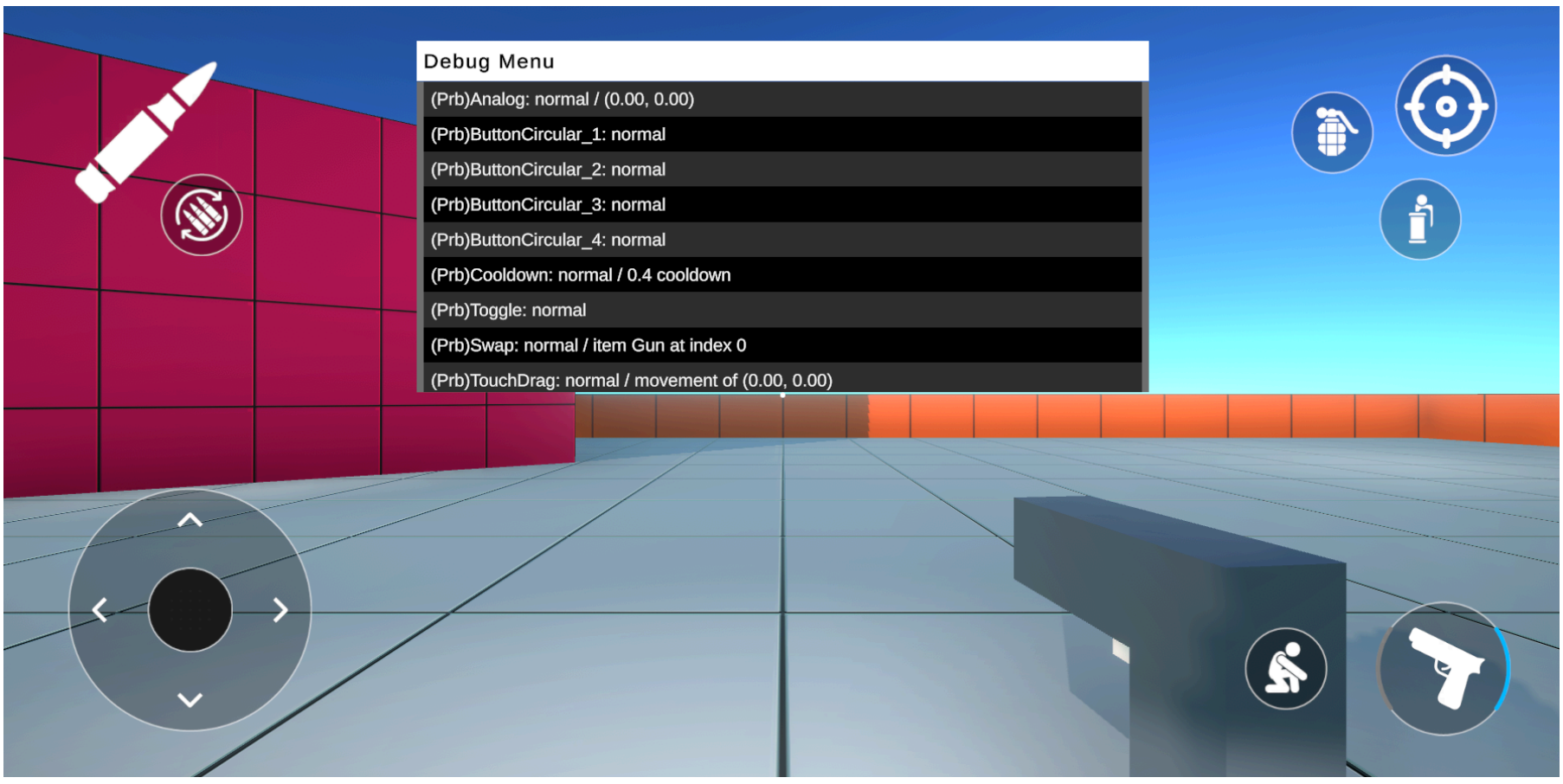


Its variables are:

- ***Display Cursor on Move***: determines if the mouse is displayed when performing a touch.
- ***Touch Rect***: reference to the Touch Rect Transform parent.

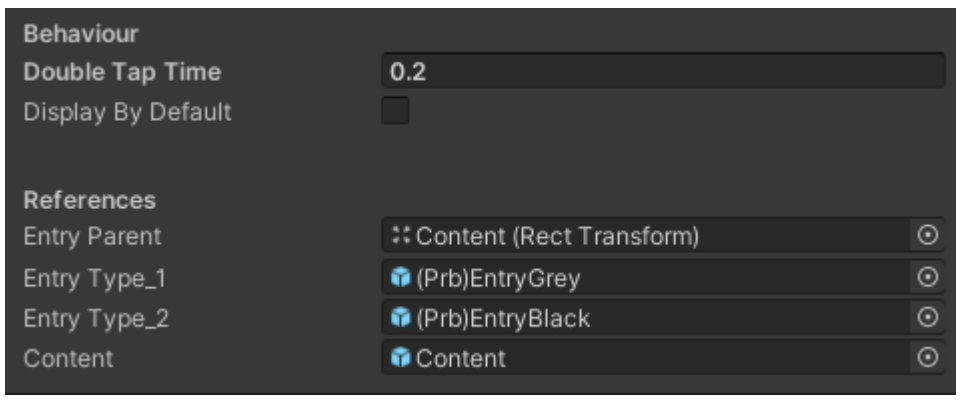
Debug Menu

The Mobile Controller Toolbox package also includes a Debug Menu that can be displayed on both Mobile and PC when a double tap is performed. The Debug Menu contains long infos on every Mobile Controller Toolbox *Input Type* currently active in the scene.



DebugMenu Component

The following is an overview of what is available in the Debug Menu Component.



Its variables are:

- ***Double Tap Time***: the time limit in which the double tap has to be performed to toggle the Debug Menu.
- ***Display By Default***: determines if the Debug Menu starting state is visible.
- ***Entry Parent***: the RectTransform containing the Debug Menu list.
- ***Entry Type 1:*** to have visual separation between elements of the Debug Menu, 2 types of prefabs are used. This is the 1st type.
- ***Entry Type 2:*** to have visual separation between elements of the Debug Menu, 2 types of prefabs are used. This is the 2nd type.
- ***Content***: the GameObject children of the Debug Menu Component.

Contact

If you found this guide useful but need further help feel free to contact me at the email ***nappin.1bit@gmail.com***
P.S. A positive review of the asset would help a lot!

Cheers