User Manual v2.1

Mobile Controller System

This asset has been designed for competitive mobile games, such as mobile MOBA or battle royale type.
The behavior of this controller is exactly the same as a successful game that exists on the market.

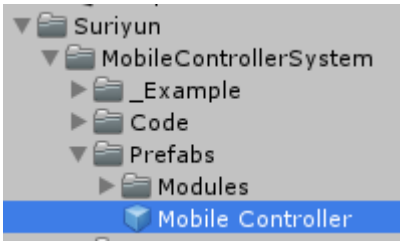
Table of Contents

Features

- Strong scripting API
- Unrestricted modification
- Smooth and accurate mobile controller system
- Mobile have same behavior as editor
- All textures are easy to replace
- Clean, easy to understand C# code
- Allow unrestricted modification
- Based on Unity UI

Quick Start : Integration

1. Drag and drop “Mobile Controller” prefab into your scene.



2. This asset requires the EventSystem to work.
If not already exist, add EventSystem to the scene by Right Click on the scene’s hierarchy > UI > Event System

---

Class UniversalButton

This is the base class of all buttons in this asset, Analog Stick also derived from this script.
Handle various touch inputs for button activation.

Class properties

Variable	Type	Remark
scaler	CanvasScaler	Reference of canvas scaler.
isAimable	bool	True = Can aim and activate. False = Activate on pressed.
btn	RectTransform	This button reference.
aimer	RectTransform	Aimer UI reference.
pointer	RectTransform	Pointer UI reference.
skillCanceller	RectTransform	Skill Cancel button reference.
hasText	bool	True if this button has a Text UI.
text	TextMeshProUGUI	Text UI of this button. Used with SetText function.
img	Image	Image component of this button.
state	ButtonState	Current state of this button.
isActive	bool	True = this button will accept touch input.
btnRadius	float	Radius of this button in pixel.
aimerRadius	float	Radius of aimer in pixels.
isManualAimOverride	bool	True when press and drag for aiming.
isFingerDown	bool	True when any finger is pressing this button.
isPointerUpOutOfBound	bool	True when pressed finger drag out of button radius.
initialFingerPosition	Vector3	First point when pressed on this button.
fingerId	int	System's finger ID that pressed on this button.
fingerPosition	Vector3	Current touch position on screen, pixel unit.
direction	Vector3	Aim direction in XY plane, value range of [-1,1]]
directionXZ	Vector3	Adjusted direction in XZ plane for easy use.
rawDir	Vector3	Raw direction value in pixel unit.
cancellerRadius	float	Cancel button radius in pixel.
horizontal	float	Return direction.x
vertical	float	Return direction.y
colorActive	Color	This button color when isActive = True.
colorInactive	Color	This button color when isActive = False;
colorPressed	Color	This button color when Pressed.
deadzoneCurve	AnimationCurve	This curve modifies input magnitude and size for deadzone.
resetOutputValueOnRelease	bool	True = When release finger after dragging the analog stick of an aimable button, output vector values will reset to Vector3.zero / False = output value retain its values after release.

**The highlighted field is probably your frequently used parameters.*

Class events. Kindly look at the handleable events from the table below.

Events	Remark
onPointerDown (int)	Int btnIndex Fired on touch.
onBeginDrag (int)	Int btnIndex After pressed, fire when start moving finger from the initial touch position.
onDrag (int)	Int btnIndex After onBeginDrag. Fired every frame the finger moved.
onPointerUp (int)	Int btnIndex Fire on release touched finger.
onEndDrag (int)	Int btnIndex After onBeginDrag. After onPointerUp. Fired on release of the finger.
onActivateSkill (int)	Int btnIndex Fired on Pressed > Release Pressed > Drag > Release Use this even along with btnIndex and aiming direction parameters to get activation position.
onCancelSkill (int)	Int btnIndex Fired on Pressed > Drag > Release on Cancel button.

Use btnIndex parameter to identify which button own the event.

Public function.

Function	Remark
SetText (string)	Set the Text UI of this button. Use this for setting skill name or set skill cooldown value.

Class AnalogStick

Extended from UniversalButton class
- Output direction and amount
- Reposition itself based on initial touch input
- Will not reposition if initial touch input is in close proximity of aiming circle
- Aiming circle will not go out of screen boundary

Events	Remark
onPointerDown (int)	Int btnIndex Fired on touch.
onBeginDrag (int)	Int btnIndex After pressed, fire when start moving finger from the initial touch position.
onDrag (int)	Int btnIndex After onBeginDrag. Fired every frame the finger moved. Use this to get real time direction value.
onPointerUp (int)	Int btnIndex Fire on release touched finger.
onEndDrag (int)	Int btnIndex After onBeginDrag. After onPointerUp. Fired on release of the finger.
onCancelSkill (int)	Int btnIndex Fired on Pressed > Drag > Release on Cancel button.

Use btnIndex parameter to identify which button own the event.

Variable	Type	Remark
dpadInner	RectTransform	Inner radius of AnalogStick if user initial touch input lands exactly on dpadInner radius. AnalogStick will position itself exactly on the initial touch position. This makes AnalogStick spawn with zero diction value as intended. If the user's initial touch lands outside of dpadInner but still within the dpadOuter area, AnalogStick will spawn with a direction value for the user wanting to move the character in that direction as intended.
dpadOuter	RectTransform	Area that can accept touch input for AnalogStick.
directionalPointer	RectTransform	Arrow UI that points to the aiming direction of AnalogStick.
dpadCosmetic	RectTransform	Appearance UI for AnalogStick.
innerRadius	float	dpadInner radius in pixel.
pointerRadius	float	directionPoint radius in pixel.

Class TouchArea

This class is mainly used for touch and drag input. Specifically handle drag input then output delta touch position in pixel and in inches with deadzone multiplier curve.

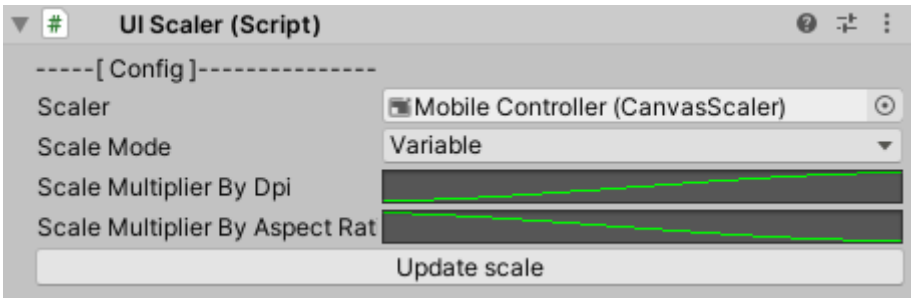
Events	Remark
onPointerDown (int)	Int btnIndex Fired on touch.
onBeginDrag (int)	Int btnIndex After pressed, fire when start moving finger from the initial touch position.
onDrag (int)	Int btnIndex After onBeginDrag. Fired every frame the finger moved. Use this to get delta finger position
onPointerUp (int)	Int btnIndex Fire on release touched finger.
onEndDrag (int)	Int btnIndex After onBeginDrag. After onPointerUp. Fired on release of the finger.
onCancelSkill (int)	Int btnIndex Fired on Pressed > Drag > Release on Cancel button.

Variable	Type	Remark
initialFingerPosition	Vector3	Initial touch position in pixel unit.
fingerPosition	Vector3	Current finger position.
deltaFingerPositionRaw	Vector3	Delta touch position since last OnDrag() event. Output in pixel unit. This won't be affected by the deadzone curve.
deltaFingerPositionInches	Vector3	Delta touch position, output in inches. Affected by deadzone curve.
deltaFingerPositionRawYX	Vector3	Swap XY parameters of deltaFingerPositionRaw for easier usage.
deltaFingerPositionInchesYX	Vector3	Swap XY parameters of deltaFingerPositionInches for easier usage.
totalDragDistance	float	Total drag distance in pixels. Reset OnEndDrag.
btnIndex	int	*Use btnIndex parameter to identify which button own the event.*

For TouchArea usage and code examples, please see Example6. It features TouchArea in Mobile FPS games.

Custom UI Scaler

This asset uses a custom UI scaler script.

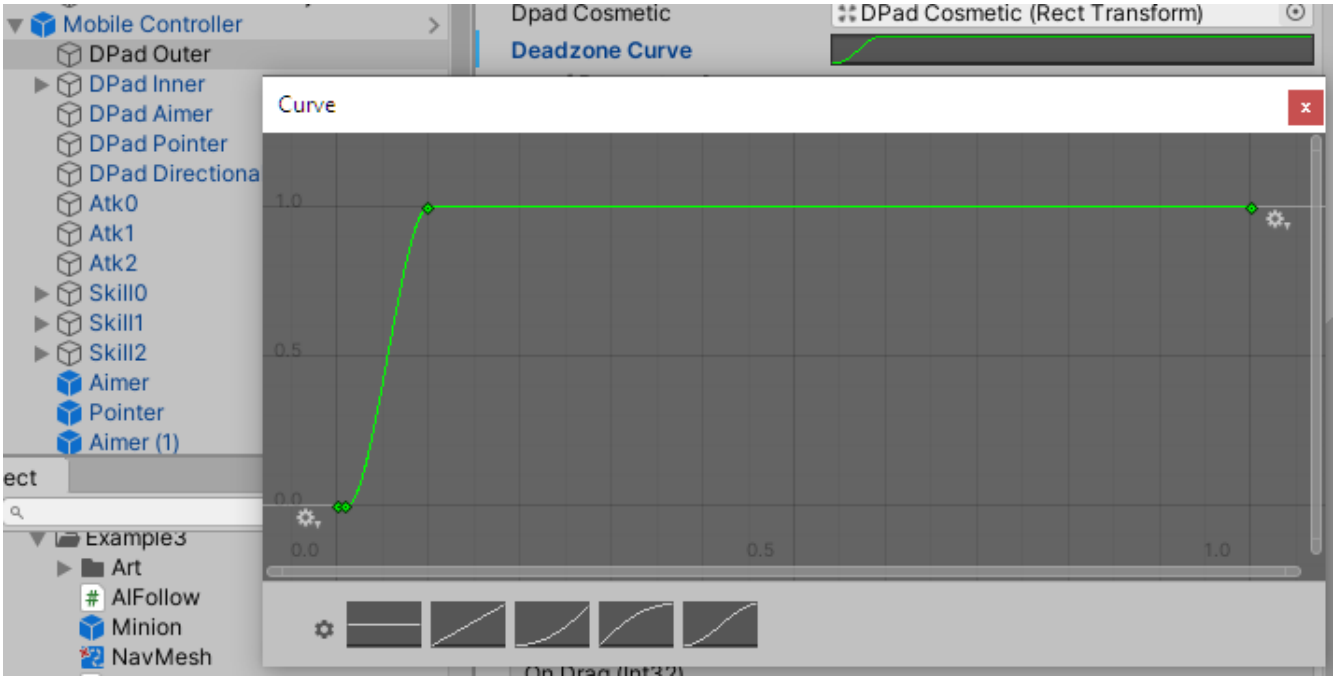


Usage : Adjust the 2 curves and test the result on actual devices.

The custom UI Scaler solved the problem for UI size across multiple screen sizes out of the box.
With our UI Scaler, developers could specify UI size for every device screen size and aspect ratio by graph UI and custom algorithm not manual labor.
And also allow developers to update their scaling parameters through API and in real-time or allow custom algorithms to be implemented.

This could be further developed into allowing users to make changes to their UI size and scale for serious pro gamers.

Deadzones Curve



The dead zone specifies how responsive your controls are when moving the analogue sticks. With a small dead zone, your position/viewpoint will move immediately when you move your analogue stick. A larger dead zone requires you to move the analogue stick farther from its point of rest in order to move the position/viewpoint on your screen.

Small Dead Zone
The advantage of a small dead zone is responsiveness. Because it requires less effort to move, you can look around and manoeuvre faster while playing. You also have more precision in making fine adjustments, such as when aiming.

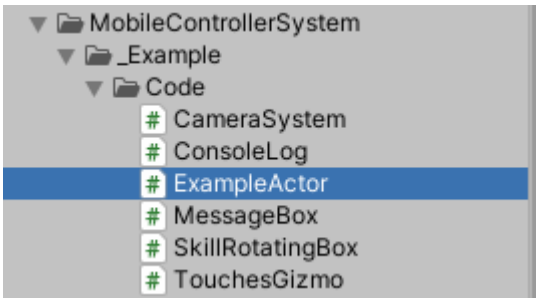
The disadvantage is that, because the controls are so sensitive, it can be hard to keep centred while you are moving. Because the game picks up any movements you make with an analogue stick, your position/viewpoint can change unexpectedly while you adjust your position if you are startled/twitchy.

Large Dead Zone
A larger dead zone eliminates unwanted movements that occur if the controls are too sensitive to your input. This means that your aim won't be spoiled if you accidentally move your analogue stick slightly once you have acquired your target. Larger dead zones also prevent the problem of ghosting (also called wandering or drifting) that can occur when analogue sticks start to wear out and cause the stick to not centre correctly. With a small dead zone, this causes the camera to spin around without your input. The downside of a large dead zone is that the controls can feel unresponsive, especially if you are used to a small dead zone.

Source : https://www.reddit.com/r/AndroidGaming/comments/b5atlq/gamepadcontroller_deadzones_psa/

Examples

Kindly take a look at ExampleActor.cs for more code examples.



This script shows how to handle various button events/parameters and will help you understand the system structures.

Example scene list

- Example : Various ways to handle MCS events.
- Example2 : Physics-based controller demo.
- Example3 : NavMesh demo.
- Example4 : 3rd Person camera with turning power curve on horizontal axis.
- Example5 : Unity's CharacterController-based. Movement direction relative to camera's rotation.
- Example6 : Mobile FPS game, TouchArea usage example.

If you have any questions kindly reach out to us via email : support@suriyun.com