using System;

using UnityEngine;

using UnityStandardAssets.CrossPlatformInput.PlatformSpecific;

namespace UnityStandardAssets.CrossPlatformInput

{

public static class CrossPlatformInputManager

{

public enum ActiveInputMethod

{

Hardware,

Touch

}

private static VirtualInput activeInput;

private static VirtualInput s\_TouchInput;

private static VirtualInput s\_HardwareInput;

static CrossPlatformInputManager()

{

s\_TouchInput = new MobileInput();

s\_HardwareInput = new StandaloneInput();

#if MOBILE\_INPUT

activeInput = s\_TouchInput;

#else

activeInput = s\_HardwareInput;

#endif

}

public static void SwitchActiveInputMethod(ActiveInputMethod activeInputMethod)

{

switch (activeInputMethod)

{

case ActiveInputMethod.Hardware:

activeInput = s\_HardwareInput;

break;

case ActiveInputMethod.Touch:

activeInput = s\_TouchInput;

break;

}

}

public static bool AxisExists(string name)

{

return activeInput.AxisExists(name);

}

public static bool ButtonExists(string name)

{

return activeInput.ButtonExists(name);

}

public static void RegisterVirtualAxis(VirtualAxis axis)

{

activeInput.RegisterVirtualAxis(axis);

}

public static void RegisterVirtualButton(VirtualButton button)

{

activeInput.RegisterVirtualButton(button);

}

public static void UnRegisterVirtualAxis(string name)

{

if (name == null)

{

throw new ArgumentNullException("name");

}

activeInput.UnRegisterVirtualAxis(name);

}

public static void UnRegisterVirtualButton(string name)

{

activeInput.UnRegisterVirtualButton(name);

}

// returns a reference to a named virtual axis if it exists otherwise null

public static VirtualAxis VirtualAxisReference(string name)

{

return activeInput.VirtualAxisReference(name);

}

// returns the platform appropriate axis for the given name

public static float GetAxis(string name)

{

return GetAxis(name, false);

}

public static float GetAxisRaw(string name)

{

return GetAxis(name, true);

}

// private function handles both types of axis (raw and not raw)

private static float GetAxis(string name, bool raw)

{

return activeInput.GetAxis(name, raw);

}

// -- Button handling --

public static bool GetButton(string name)

{

return activeInput.GetButton(name);

}

public static bool GetButtonDown(string name)

{

return activeInput.GetButtonDown(name);

}

public static bool GetButtonUp(string name)

{

return activeInput.GetButtonUp(name);

}

public static void SetButtonDown(string name)

{

activeInput.SetButtonDown(name);

}

public static void SetButtonUp(string name)

{

activeInput.SetButtonUp(name);

}

public static void SetAxisPositive(string name)

{

activeInput.SetAxisPositive(name);

}

public static void SetAxisNegative(string name)

{

activeInput.SetAxisNegative(name);

}

public static void SetAxisZero(string name)

{

activeInput.SetAxisZero(name);

}

public static void SetAxis(string name, float value)

{

activeInput.SetAxis(name, value);

}

public static Vector3 mousePosition

{

get { return activeInput.MousePosition(); }

}

public static void SetVirtualMousePositionX(float f)

{

activeInput.SetVirtualMousePositionX(f);

}

public static void SetVirtualMousePositionY(float f)

{

activeInput.SetVirtualMousePositionY(f);

}

public static void SetVirtualMousePositionZ(float f)

{

activeInput.SetVirtualMousePositionZ(f);

}

// virtual axis and button classes - applies to mobile input

// Can be mapped to touch joysticks, tilt, gyro, etc, depending on desired implementation.

// Could also be implemented by other input devices - kinect, electronic sensors, etc

public class VirtualAxis

{

public string name { get; private set; }

private float m\_Value;

public bool matchWithInputManager { get; private set; }

public VirtualAxis(string name)

: this(name, true)

{

}

public VirtualAxis(string name, bool matchToInputSettings)

{

this.name = name;

matchWithInputManager = matchToInputSettings;

}

// removes an axes from the cross platform input system

public void Remove()

{

UnRegisterVirtualAxis(name);

}

// a controller gameobject (eg. a virtual thumbstick) should update this class

public void Update(float value)

{

m\_Value = value;

}

public float GetValue

{

get { return m\_Value; }

}

public float GetValueRaw

{

get { return m\_Value; }

}

}

// a controller gameobject (eg. a virtual GUI button) should call the

// 'pressed' function of this class. Other objects can then read the

// Get/Down/Up state of this button.

public class VirtualButton

{

public string name { get; private set; }

public bool matchWithInputManager { get; private set; }

private int m\_LastPressedFrame = -5;

private int m\_ReleasedFrame = -5;

private bool m\_Pressed;

public VirtualButton(string name)

: this(name, true)

{

}

public VirtualButton(string name, bool matchToInputSettings)

{

this.name = name;

matchWithInputManager = matchToInputSettings;

}

// A controller gameobject should call this function when the button is pressed down

public void Pressed()

{

if (m\_Pressed)

{

return;

}

m\_Pressed = true;

m\_LastPressedFrame = Time.frameCount;

}

// A controller gameobject should call this function when the button is released

public void Released()

{

m\_Pressed = false;

m\_ReleasedFrame = Time.frameCount;

}

// the controller gameobject should call Remove when the button is destroyed or disabled

public void Remove()

{

UnRegisterVirtualButton(name);

}

// these are the states of the button which can be read via the cross platform input system

public bool GetButton

{

get { return m\_Pressed; }

}

public bool GetButtonDown

{

get

{

return m\_LastPressedFrame - Time.frameCount == -1;

}

}

public bool GetButtonUp

{

get

{

return (m\_ReleasedFrame == Time.frameCount - 1);

}

}

}

}

}