Controller Mapping

Input System

Unity Input System is a new way to have consistent controls across a wide range of devices. Unity Input System allows easy customization for multiplayer input and on the fly switching to different controllers.
While Unity supports a wide range of controllers, It is dependent on SDL for Input on Linux OS. While many controllers are already known and recognised by SDL, Atari controllers are yet to be recognised.

1. Unlike Windows and MacOs,In Linux, Unity asks SDL for information about any device.SDL then matches the vendor id and device id with its database and returns a layout with the best match possible.
2. Layout is a map of all the buttons present in a controller. If Unity cannot find that map, it will not be able to register any input from that device.
3. SDL doesn't recognise Atari controllers, so it cannot provide a perfect layout for these controllers, although it does return a device with the same device id which we can then use to extend that device's layout on runtime.Unity allows a developer to extend a layout to include more buttons.
4. Modern and Classic Controllers are received by SDL in a Four CC Format (ie. LJOY)
5. Modern Controller Device version is 1002
6. Classic Joystick Device version is 1001
7. If you look into SDL, these Device ids are matched as Keyboard Hub and Cetus CDC Device.So, For eg. Whenever a modern controller is connected, Unity will notify the developer that a device "Keyboard Hub" is connected. Developers can then on the fly extend that layout to have all the modern controller buttons.

8. Check Unity's Documentation for extending a layout for matched devices.

Please refer to the bits and offset values of each button of modern controller and classic joystick

9. Once a developer is able to set up the layout for both classic and modern controllers, They shall proceed to set up the Action map for their Game Controls. An Action Map as the name suggests is a keybinding to a certain action

10. Unity Tutorial: https://youtu.be/xF2zUOfPyg8

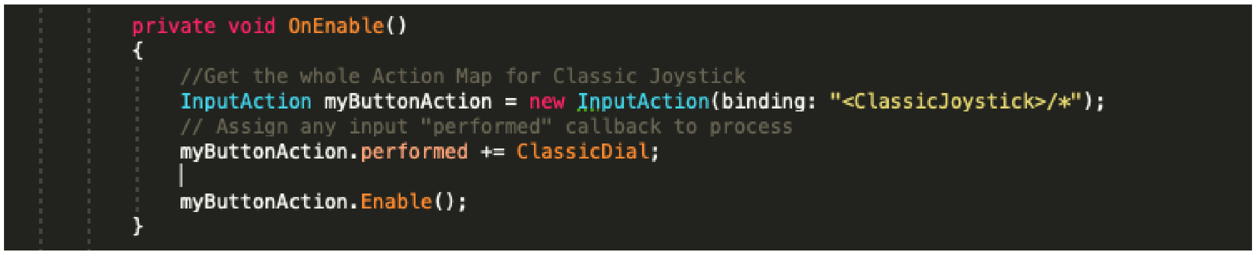
Modern Controller

Button	Bits	Offset
A	0	0
B	1	0
X	3	0
Y	2	0
Back	6	0
Atari	0	1
Menu	7	0
Left Joystick Horizontal	0	4
Left Joystick Vertical	0	8
Right Joystick Horizontal	0	16
Right Joystick Vertical	0	20
D-Pad Horizontal	0	28
D-Pad Vertical	0	32
Left Bumper	4	0
Right Bumper	5	0
Left Trigger	0	12
Right Trigger	0	24
Left Joystick Press	1	1
Right Joystick Press	2	1

Classic Joystick

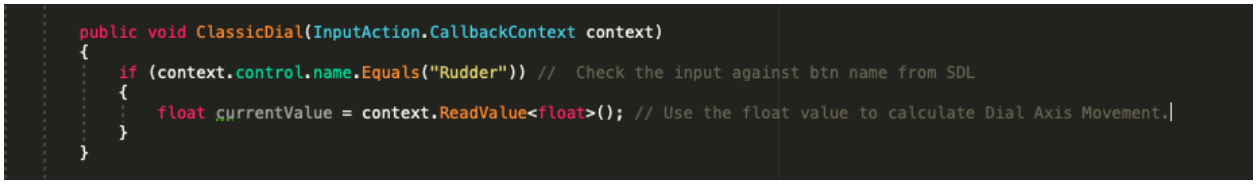
Button	Bits	Offset
A	0	0
B	1	0
Back	2	0
Atari	4	0
Menu	3	0
Joystick Horizontal	0	8
Joystick Vertical	0	12

*To read Classic Controller Joystick Twist, use the following code -



```
private void OnEnable()
{
    //Get the whole Action Map for Classic Joystick
    InputAction myButtonAction = new InputAction(binding: "<ClassicJoystick>/*");
    // Assign any input "performed" callback to process
    myButtonAction.performed += ClassicDial;

    myButtonAction.Enable();
}
```



```
public void ClassicDial(InputAction.CallbackContext context)
{
    if (context.control.name.Equals("Rudder")) //  Check the input against btn name from SDL
    {
        float currentValue = context.ReadValue<float>(); // Use the float value to calculate Dial Axis Movement.
    }
}
```

Input Manager

Modern Controller

Button	**Joystick Button/Axis**
A	joystick button 0
B	joystick button 1
X	joystick button 3
Y	joystick button 2
Back	joystick button 6
Atari	joystick button 8
Menu	joystick button 7
Left Joystick Horizontal	Axis 1
Left Joystick Vertical	Axis 2
Right Joystick Horizontal	Axis 4
Right Joystick Vertical	Axis 5
D-Pad Horizontal	Axis 7
D-Pad Vertical	Axis 8
Left Bumper	joystick button 4
Right Bumper	joystick button 5
Left Trigger	Axis 3
Right Trigger	Axis 6
Left Joystick Press	joystick button 9
Right Joystick Press	joystick button 10

Classic Joystick

Button	**Joystick Button/Axis**
A	joystick button 0
B	joystick button 1
Back	joystick button 2
Atari	joystick button 4
Menu	joystick button 3
Joystick Horizontal	Axis 2
Joystick Vertical	Axis 3
Joystick Rotation/Twist	Axis 1