Using the new Input System

Overview

With the release of Unity 6, the new Input System, simply referred to as the Input System, is now the standard for managing player input in Unity projects. While the Legacy Input System has served developers well for many years by providing robust and reliable input handling across numerous platforms, the new Input System offers enhanced flexibility and better support for modern devices and platforms. The Input System simplifies the process of setting up, configuring, and managing player input through code, making it easier to develop and iterate on user input systems. As of Unity 6, developers are encouraged to use the Input System for all new projects to take full advantage of its advanced features and capabilities.

In the solution video for this session, you saw how to set up the player controller using the Legacy Input System. While this approach to player input is still supported in Unity 6, the certification exam will test you on your knowledge of the current Input System. Your extra challenge for this session will be to create a version of the controller using the current system. This tutorial will walk you through the process of upgrading your project to support both Input Systems, and prepare you for the session challenge.

Setup for Projects upgraded to Unity 6

In new Unity 6 projects, the Input System is installed by default. However, if you are upgrading a project from an earlier version of Unity, you will need to install the Input System package from the Package Manager. Older projects must also ensure that the project's API compatibility level is set to .Net Framework. While the project you were given is in Unity 6, it was upgraded from 2019 LTS, and this process was not completed for you.

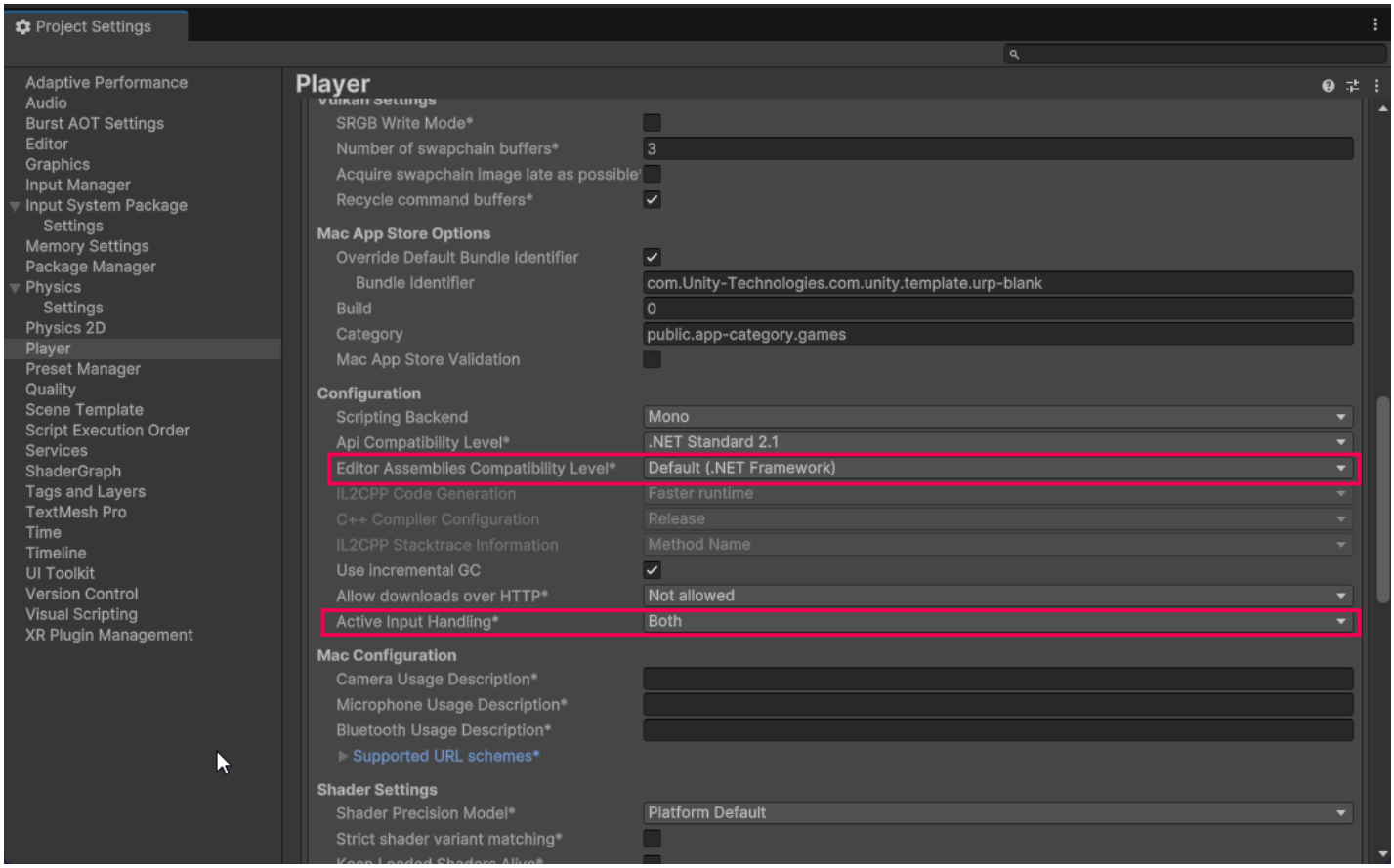
To install the Input System from the Package Manager and upgrade the API compatibility level for upgraded projects:

1. Navigate to **Edit** > **Project Settings** to open up the Project Settings window.

2. Select the Player option from the left column, and then select the triangle to expand the options for Other Settings in the main Project Settings window.

3. Scroll down to the **Configuration** option, and ensure that the setting for **API Compatibility Level** field is set to **Default (.Net Framework)** .

4. Set the Active Input Handling option to Both.



5. A popup will appear and notify you that the Unity editor must be restarted for the change to take effect. Select Apply and let the project restart, and once it has, close the Project Settings window.

6. Navigate to the Package Manager by selecting **Window > Package Manager** from the top menu.

7. On the left side of the Package Manager window, select Unity Registry, and locate the Input System from the list. Select Install.

The Input System is now configured for your upgraded project.

Create an Input Action

In your extra challenge, your task will be to set up a Prefab Variant of the Player Controller using the new Input System. To accomplish this, you'll create your Variant here and add an Input Action.

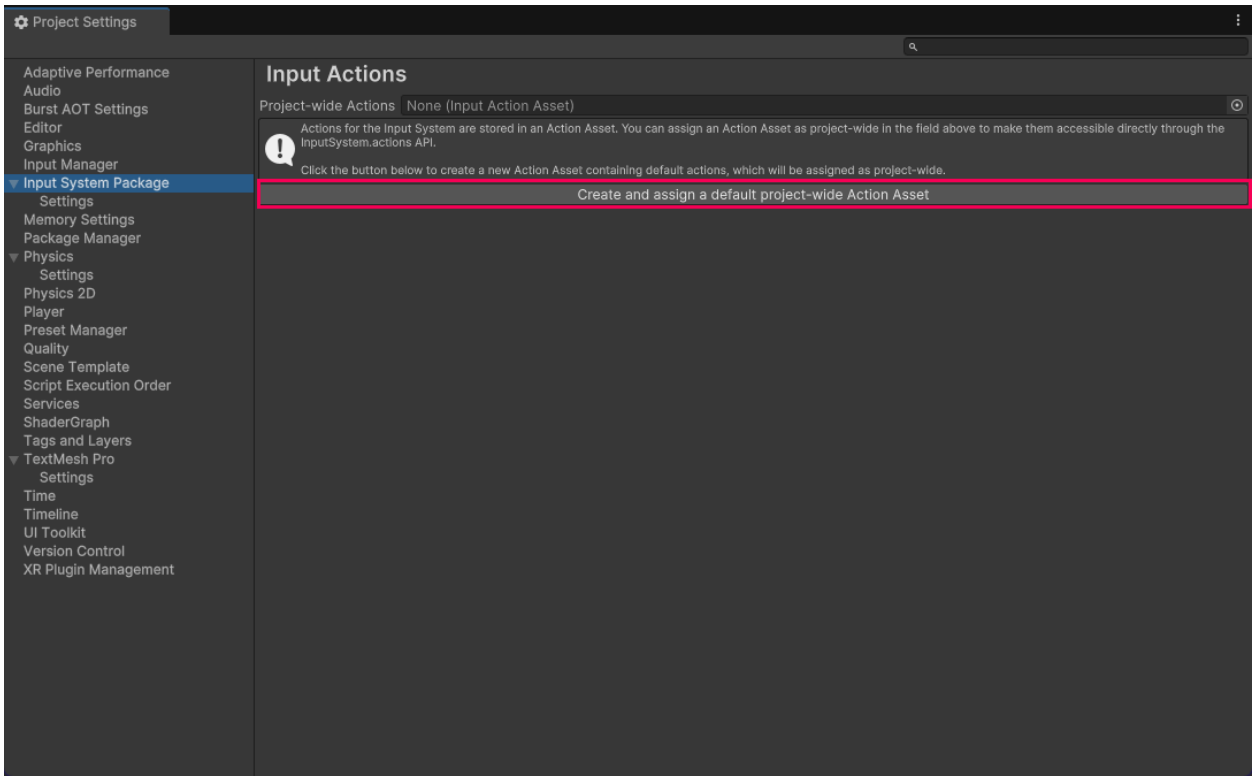
1. Create a Prefab Variant of your Player Controller.

2. Remove the existing Player Controller script from the Variant.

For projects upgraded to Unity 6, a default Input Action asset must be created. Projects created with Unity 6 contain this asset by default.

To create a default Input Action asset:

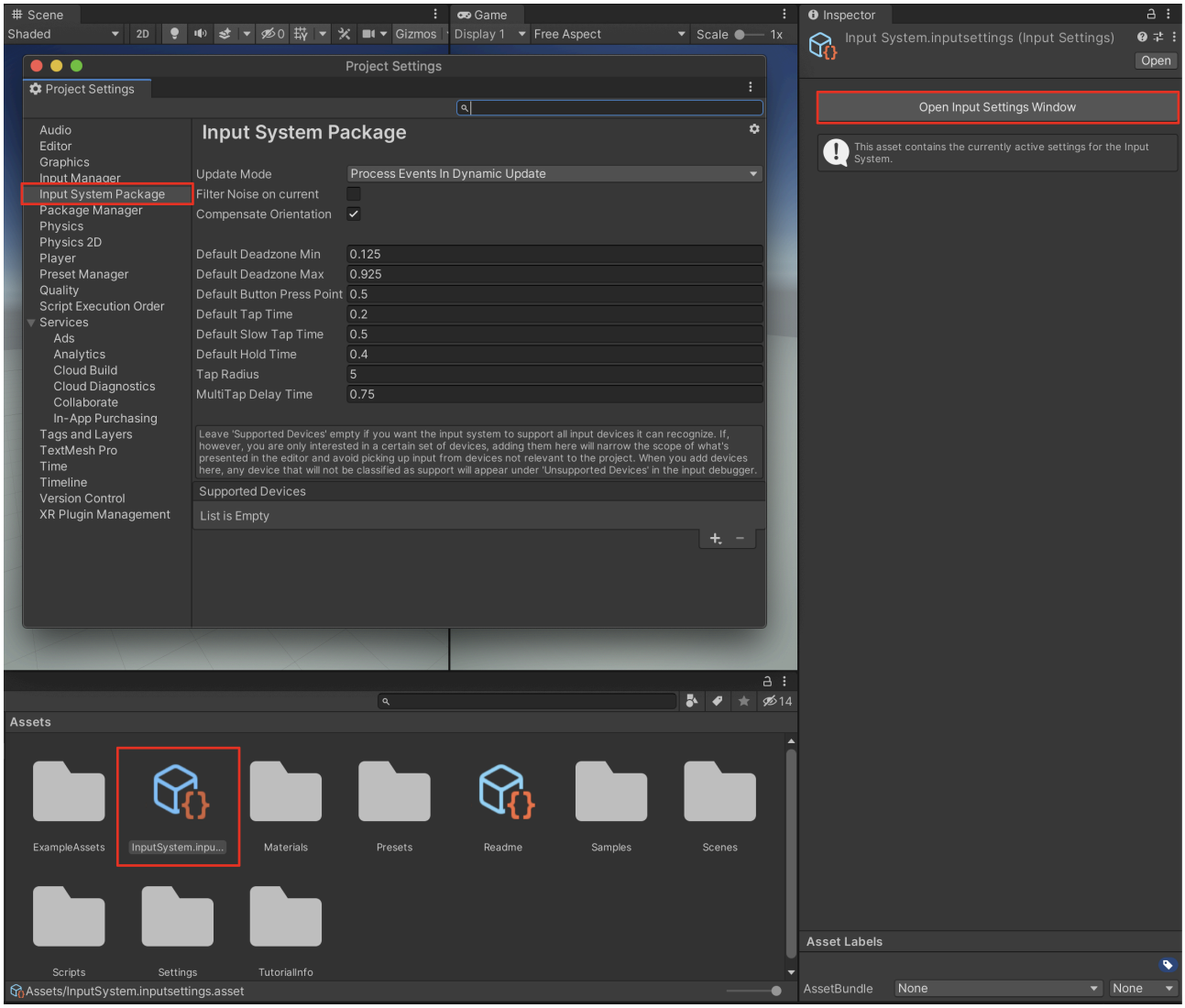
3. Navigate to **Edit > Project Settings** and select the **Input System Package** from the left column of the Project Settings window. Select the **Create and assign a default project-wide Action asset** button.



A newly created input settings asset is now saved in the root of your Project folder structure as InputSystem_Actions. You are able to customize settings that pertain to how events are processed in either update or fixed update methods, and the default timings of button taps.

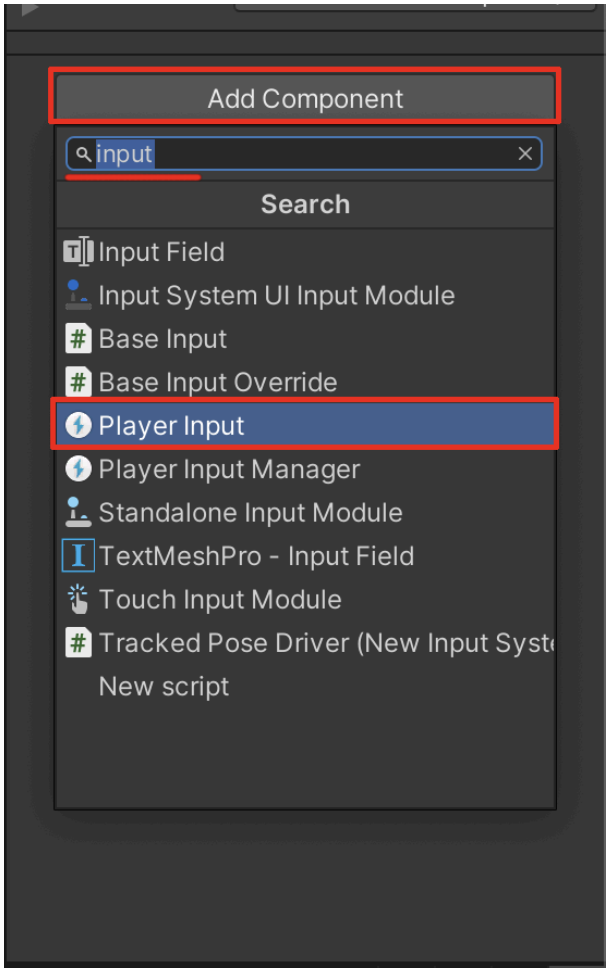
4. Leave the Input System Package settings unchanged with their default settings for now, and close the Project Settings window.

To edit any of these settings later on, you can either open up the Project Settings window again, or first select the InputSystem asset in the Project window, and then select the **Open Input Settings Window** button in the Inspector to open the Project Settings window.

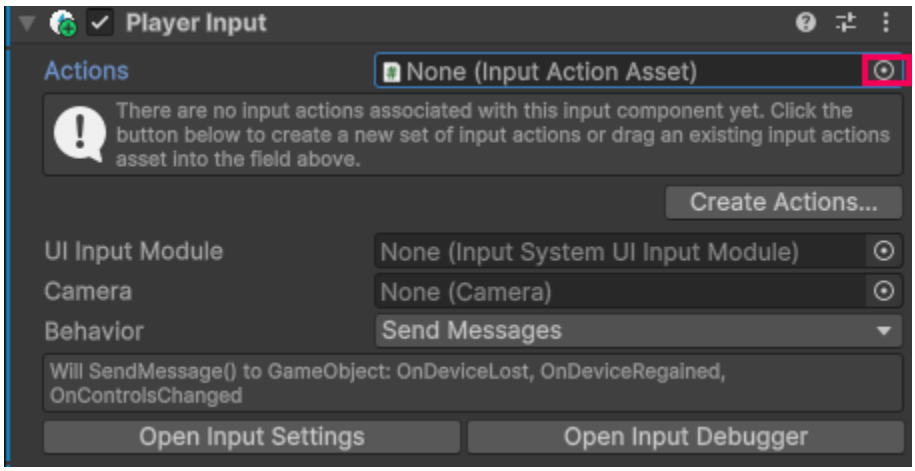


In order to control the player's movements, you must first add the Player input component.

5. Select your player GameObject in the Hierarchy window. Next, select the **Add Component** button in the Inspector, type "player" in the search field, and then select **Player Input** from the drop-down menu.



6. The InputSystem_Actions that you created previously should be automatically added to the Actions input parameter. If it is empty, select the circle icon to open the selection window, and select your InputSystem_Actions asset.



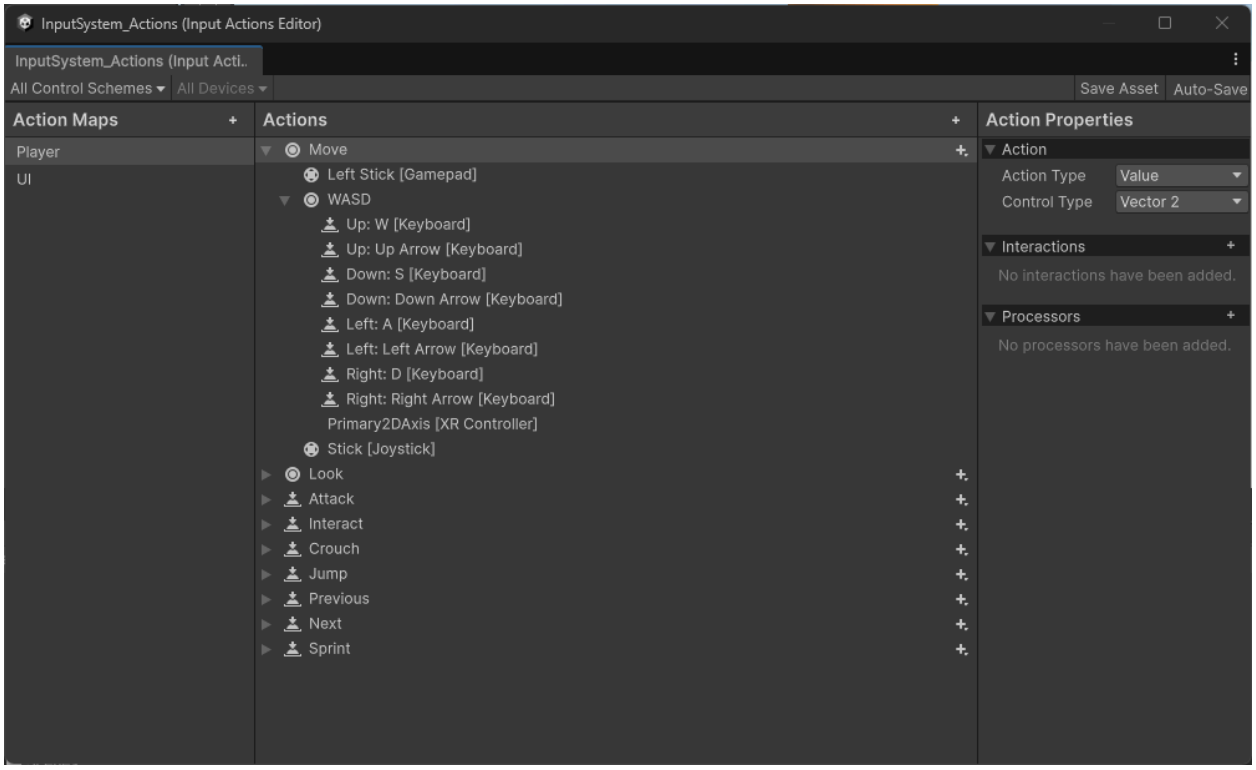
In the next step, you'll use the Player Input component to send messages to your script when keyboard input is detected. To read more about Player Input, refer to the user manual page.

Your player is now fully configured and ready to use the Input System.

Use the Input Action asset to receive player input

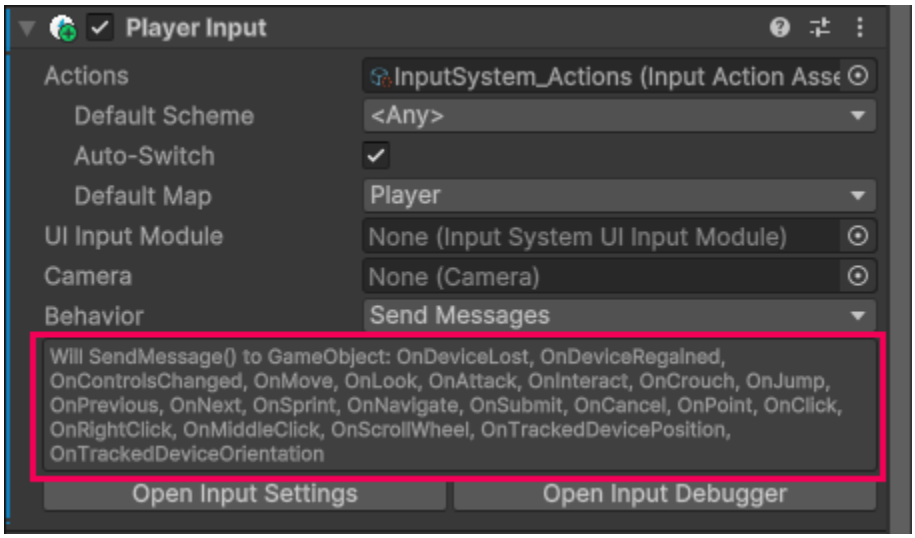
The Input System actions come preconfigured with common user inputs for games. This means that rather than having to program input detection yourself, you need only write the functionality for what should occur once the input has been received.

To view all of the preconfigured inputs, double click the InputSystem_Actions asset in your Project to open the Input Actions Editor.



Notice that movement detection is already fully configured for keyboard controls, in addition to supporting game pads, virtual joysticks (such as for mobile applications) and even XR controllers. This means that after creating your new movement controls for the keyboard, if you plugged in an Xbox controller, you would be able to control your game without any additional programming.

By default, the Player Input component's Behavior is set to Send Messages whenever specific inputs are made. Once an Input Action is added to Player Inputs, you should see a large selection of messages that could be sent:



To receive basic keyboard input, you'll use the `OnMove` method.

1. Create a new C# script and give it an appropriate name to differentiate it from the legacy input controller.

2. Attach the new script to your Variant Prefab, then open the script in your IDE.

3. Add the input system namespace to your script.

```
using UnityEngine.InputSystem;
```

4. Create a Rigidbody variable and initialize it in the `Start` method.

```
public class AlternatePlayerInput : MonoBehaviour
{
    private Rigidbody rb;

    void Start()
    {
        rb = GetComponent<Rigidbody>();
    }
```

5. Create three float variables to store the X and Y input axes and the speed in which you want your player to move. Name them `movementX, movementY and speed`. Initialize speed to a value of your choosing:

```
private float movementX;
private float movementY;
public float speed = 50;
```

Note: Float values for speed below 15 will be difficult to see while playtesting. It's recommended to have high values of 20 or above to start.

6. Create the OnMove method. Because this method is based on Input, an InputValue must be passed as a parameter.

```
private void OnMove(InputValue movementValue)
{
    Vector2 movementVector = movementValue.Get<Vector2>();
    movementX = movementVector.x;
    movementY = movementVector.y;
}
```

Whenever the Player Input component detects user input that correlates to movement as defined in Input Actions, `OnMove` will be called. The input value will be passed to a temporary Vector 2 variable, which is then split out and assigned to the `movementX` and `movementY` variables. This gives you access to the horizontal and vertical movement axes, which can be used to apply force to the Rigidbody.

7. Replace the Update method with FixedUpdate to account for the usage of the physics system via the Rigidbody. Use the movement floats to create a new Vector3 value for use with the Rigidbody:

```
void FixedUpdate()
{
    Vector3 movement = new Vector3(movementX, 0.0f, movementY);
    rb.AddForce(movement * speed);
}
```

8. Save the script, and switch back to Unity to playtest the controller using the WASD keys.

Your player should now move based on keyboard input.

Further information about the Input System can be found in the Input System Manual.