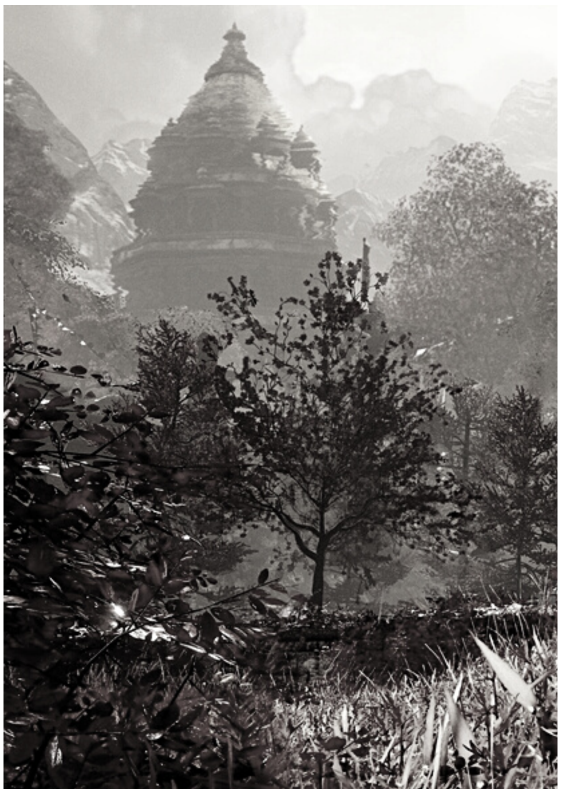
NYX

Startup guide & documentation

Jay Morrigton

SHARK.studio

2022-2023



SUMMARY

INSTALLATION

Install NYX like every assets by accessing:
Windows>Package-Manager>My-Assets
And then searching for NYX.

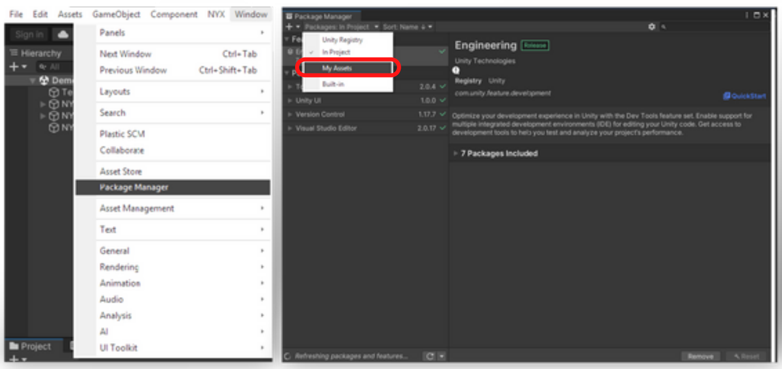
Then, make sure you are using
the old-input manager into your project.

Now, you can replace your default Unity InputManager settings with the template provided by NYX, this step is not obligatory but since the defaults axis will not be used (except for UI navigation & mouse control) we can delete them, you can do it manually of course but the preset provided by NYX keep just the necessary axis for the UI and delete the rest for you.

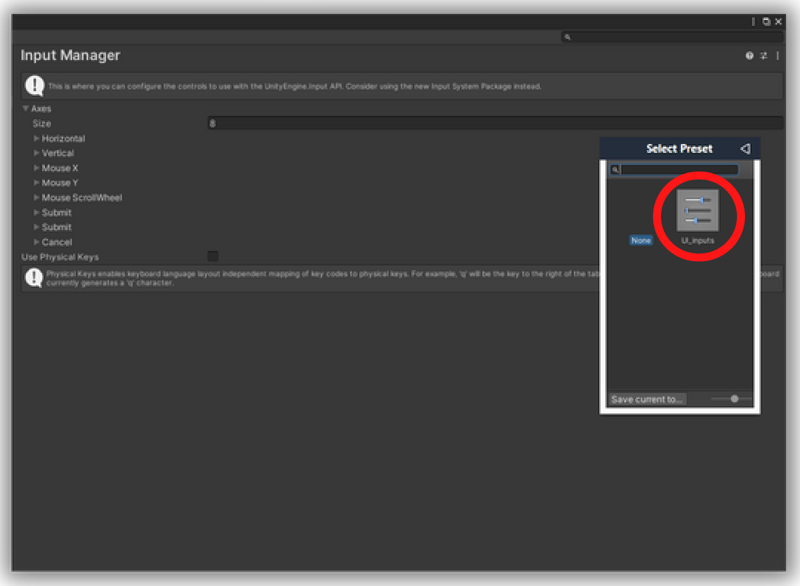
You are now ready to use NYX,
you can open the test-scene and verify that you don't get any error in the console.

NOTICE : Pay attention to NEVER rename any of the NYX folders !

If you get in trouble, you can always ask for somes help on our discord here : {LINK}
We will be glad to help you :)



Optimize your development experience in Unity with the Dev Tools feature set. Enable support for multiple integrated development environments (IDE) for editing your Unity code. Get access to development tools to help you test and analyze your project's performance.



This is where you can configure the controls to use with the UnityEngine.Input API. Consider using the new Input System Package instead.

Physical Keys enables keyboard language layout independent mapping of key codes to physical keys. For example, 'q' will be the key to the right of the tab ... board currently generates a 'q' character.

GET STARTED

After opening the NYX demo-scene,
you will see two main GameObjects :

- NYX_Systems
- NYX_GUI

NYX_Systems contain the InputManager, which is the core of the system & also a InputDemos that act as a way to display your axis & keys in real-time.

NYX_GUI contain the full in-game input remap system with a ready to use UI.

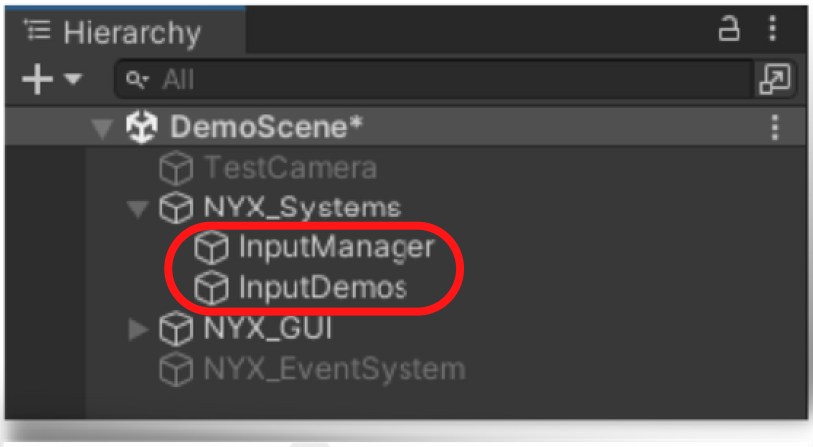
Both of theses GameObjects can be added to your scene by accessing :
NYX > Scene-Management window on the context menu at the top of the editor.

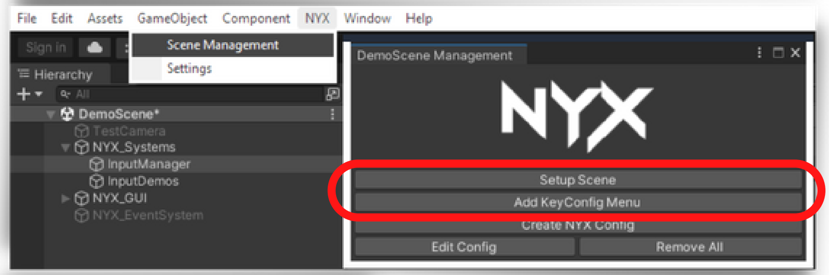
If you select the InputManager, you should see in the inspector two tabs appearing :

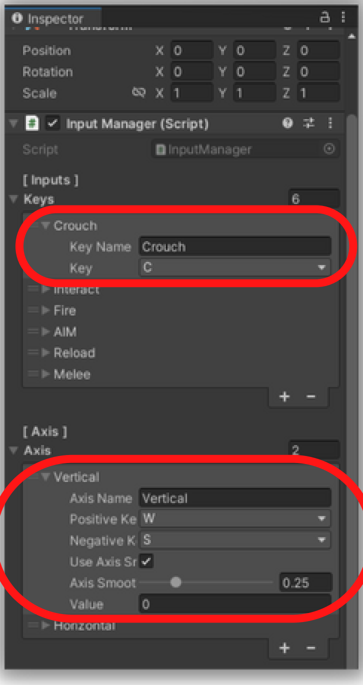
- Keys
- Axis

Keys are actions that can be assigned to a single key (for example : [R] to reload).

Axis works the same as axis in the unity-editor, except that they have more features, like editing in real time, axis smoothing and a save & load system.







Position	X 0	Y 0	Z 0
Rotation	X 0	Y 0	Z 0
Scale	X 1	Y 1	Z 1

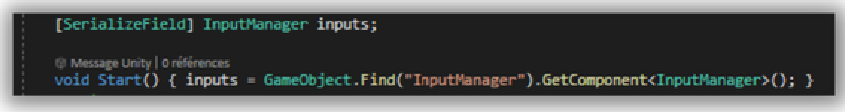
Script: InputManager

[Inputs]
Keys: 6
- Crouch
  - Key Name: Crouch
  - Key: C
- Interact
- Fire
- AIM
- Reload
- Melee

[Axis]
Axis: 2
- Vertical
  - Axis Name: Vertical
  - Positive Ke: W
  - Negative K: S
  - Use Axis Sr: ✓
  - Axis Smoot: 0.25
  - Value: 0
- Horizontal

USE THE TOOL

To use the tool for your Player inputs, you will first need to make a reference to the "InputManager" script.
You can do it with two different ways:



```
[SerializeField] InputManager inputs;

Message Unity | 0 références
void Start() { inputs = GameObject.Find("InputManager").GetComponent<InputManager>(); }
```

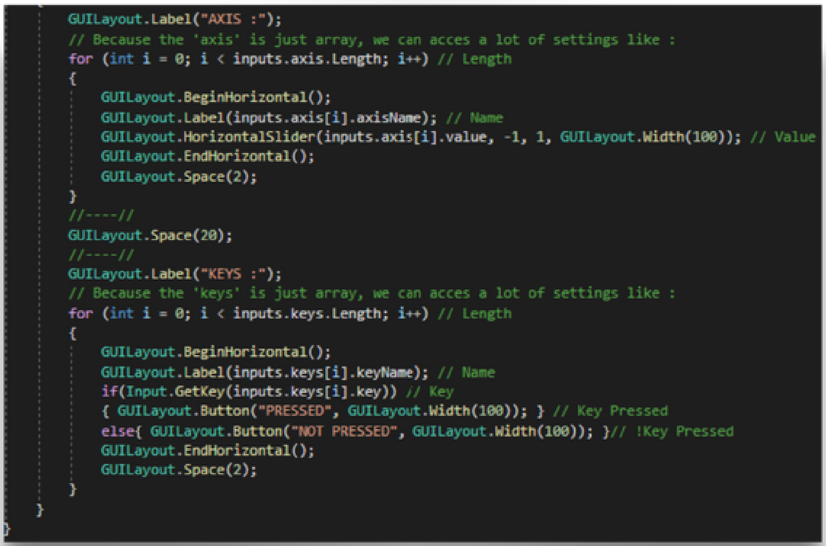
- by using a var to fill in the inspector :

[SerializeField] InputManager inputs;

- or by asigning at start :

inputs = GameObject.Find("InputManager");

then you can get the value of an axis with the index of this axis :inputs.axis[i].value
or get the keycode with the same process : inputs.keys[i].key



```
    GUILayout.Label("AXIS :");
    // Because the 'axis' is just array, we can acces a lot of settings like :
    for (int i = 0; i < inputs.axis.Length; i++) // Length
    {
        GUILayout.BeginHorizontal();
        GUILayout.Label(inputs.axis[i].axisName); // Name
        GUILayout.HorizontalSlider(inputs.axis[i].value, -1, 1, GUILayout.Width(100)); // Value
        GUILayout.EndHorizontal();
        GUILayout.Space(2);
    }
    //----//
    GUILayout.Space(20);
    //----//
    GUILayout.Label("KEYS :");
    // Because the 'keys' is just array, we can acces a lot of settings like :
    for (int i = 0; i < inputs.keys.Length; i++) // Length
    {
        GUILayout.BeginHorizontal();
        GUILayout.Label(inputs.keys[i].keyName); // Name
        if(Input.GetKey(inputs.keys[i].key)) // Key
        { GUILayout.Button("PRESSED", GUILayout.Width(100)); } // Key Pressed
        else{ GUILayout.Button("NOT PRESSED", GUILayout.Width(100)); }// !Key Pressed
        GUILayout.EndHorizontal();
        GUILayout.Space(2);
    }
  }
}
```

You can also get others exposed vars :

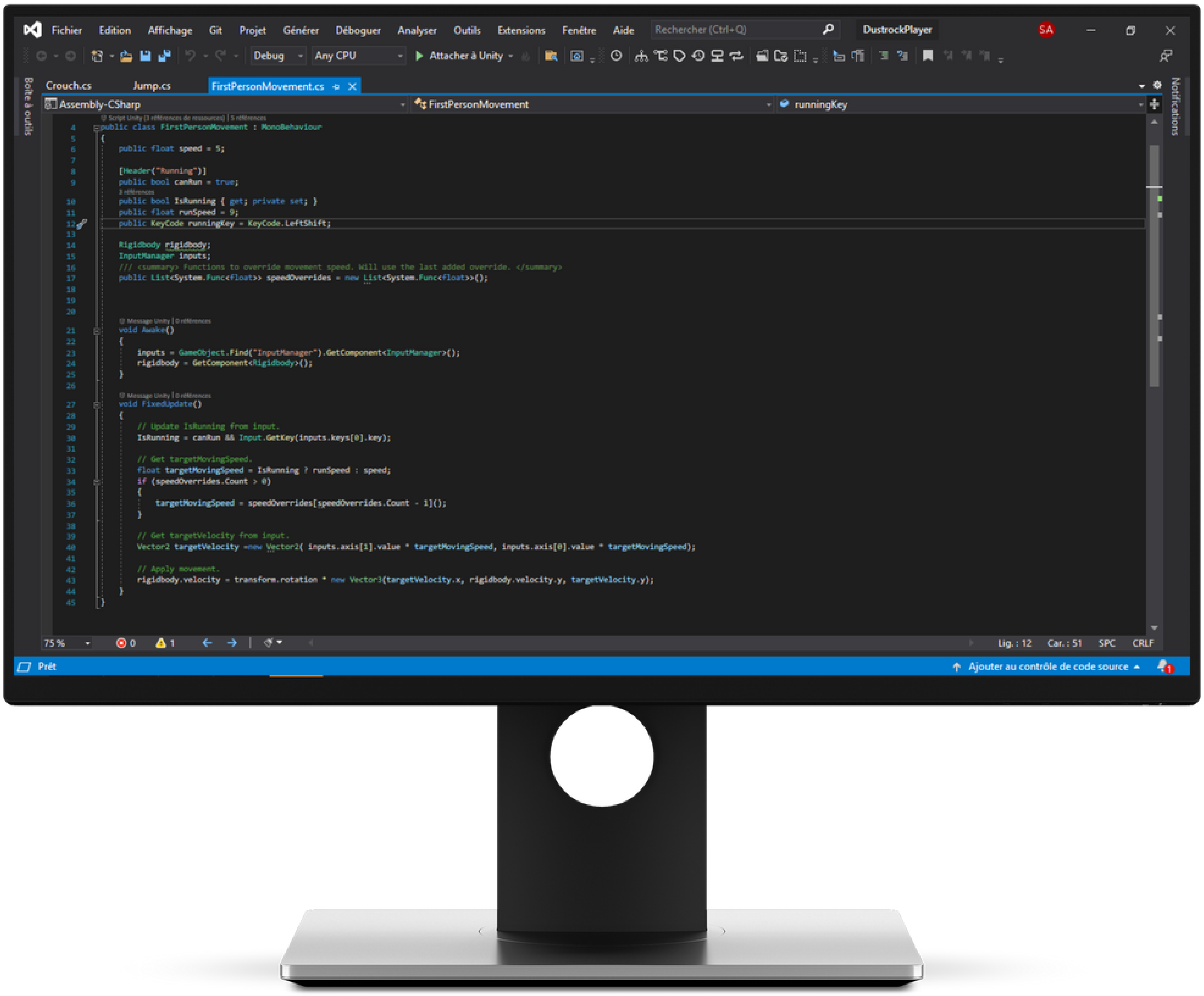
- (string) name / axisName
- (float) axisSmoothingAmount
- (bool) useAxisSmoothing
- (keycode) negativeKey
- (keycode) positiveKey
- (float) ref

Just use theses vars to control your player or other interactions in your game.

The rest (saving, loading, optimisation, etc...) is automatically setup for you.

EXAMPLE

Here is an example of usage on the
Mini First Person Controller
by @Simon Pasi



```csharp
public class FirstPersonMovement : MonoBehaviour
{
    public float speed = 5;

    [Header("Running")]
    public bool canRun = true;
    public bool IsRunning { get; private set; }
    public float runSpeed = 9;
    public KeyCode runningKey = KeyCode.LeftShift;

    Rigidbody rigidbody;
    InputManager inputs;
    /// <summary> Functions to override movement speed. Will use the last added override. </summary>
    public List<System.Func<float>> speedOverrides = new List<System.Func<float>>();



    void Awake()
    {
        inputs = GameObject.Find("InputManager").GetComponent<InputManager>();
        rigidbody = GetComponent<Rigidbody>();
    }

    void FixedUpdate()
    {
        // Update IsRunning from input.
        IsRunning = canRun && Input.GetKey(inputs.keys[0].key);

        // Get targetMovingSpeed.
        float targetMovingSpeed = IsRunning ? runSpeed : speed;
        if (speedOverrides.Count > 0)
        {
            targetMovingSpeed = speedOverrides[speedOverrides.Count - 1]();
        }

        // Get targetVelocity from input.
        Vector2 targetVelocity =new Vector2( inputs.axis[1].value * targetMovingSpeed, inputs.axis[0].value * targetMovingSpeed);

        // Apply movement.
        rigidbody.velocity = transform.rotation * new Vector3(targetVelocity.x, rigidbody.velocity.y, targetVelocity.y);
    }
}
```

2022-2023

THANKS FOR USING

SHARK

STUDIO