Simple Motocross Physics

Documentation

Offered by AIKODEX

Simple Motocross Physics package for Unity delivers a joint physics-controlled approach for motorbike simulation. This asset requires Animation Rigging v1.0.0+. Native 1.0.3. Inclusion of Animation Rigging package allows for an IK framework which allows a bipedal character avatar to ride any pedaled vehicle with appropriate posture and mechanics.

The Unity3D joint physics, controlled and corrected by quaternions, allows for a robust and majorly unbreakable physics system. The inputs are based on axes hence compatible with all platforms, including but not limited to, Windows (PC), Mac, Universal Windows platform (UWP), Linux Standalone, iOS, Android, ARKit, ARCore, Microsoft HoloLens, Windows Mixed Reality, Magic Leap (Lumin), Oculus, PlayStation VR, PS5, PS4, Xbox One, Xbox X|S, Nintendo Switch, Google Stadia, WebGL.

This package supports all render pipelines: Standard, High-Definition Render Pipeline (HDRP), Universal Render Pipeline (URP) and Scriptable Render Pipeline (SRP).

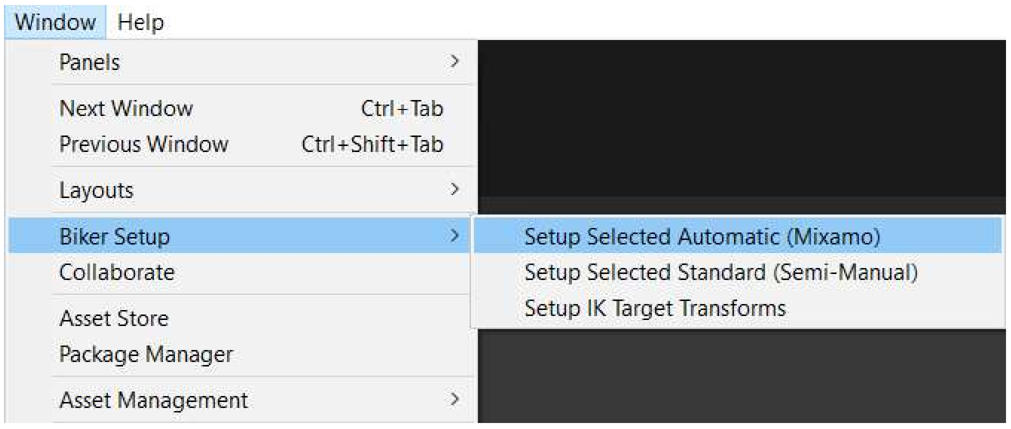
DETAILS

- Animations: Contains the Avatar controller for the biker and a few basic animations like idle, transition to riding and reversing. The included Unity's mechanim based animations are of humanoid type and apply to all humanoid characters.

- Editor: Contains a script which sets up the selected custom character in the hierarchy to ride the bike.

  Setup Selected Automatic (Mixamo)
  To set up a custom mixamo character, select the GameObject with the moto controller script attached and drag and drop your character into the hierarchy. Expand the hierarchy and scroll down to find the character. Select the character.

  With the character selected, click on Setup Selected.

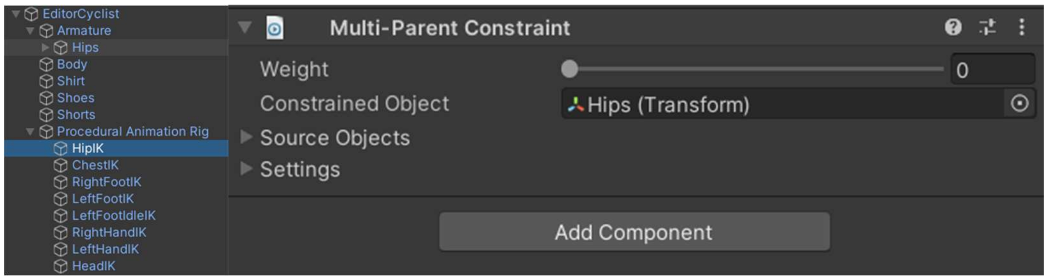

Window
Help
Panels
Next Window Ctrl+Tab
Previous Window Ctrl+Shift+Tab
Layouts
Biker Setup
Collaborate
Asset Store
Package Manager
Asset Management
Setup Selected Automatic (Mixamo)
Setup Selected Standard (Semi-Manual)
Setup IK Target Transforms

  This will transfer the properties of the biker such as Transform, Animator Controller, Procedural IK Handler to your custom character in one click.

  Delete the original biker.

Setup Selected Standard (Semi-Manual)

Repeat the same process but go through the Procedural Rig's gameObjects and fill in the inspector values with your custom character's armature parts.



- EditorCyclist
  - Armature
    - Hips
  - Body
  - Shirt
  - Shoes
  - Shorts
  - Procedural Animation Rig
    - HipIK
    - ChestIK
    - RightFootIK
    - LeftFootIK
    - LeftFootIdleIK
    - RightHandIK
    - LeftHandIK
    - HeadIK

Multi-Parent Constraint

Weight 0

Constrained Object Hips (Transform)

Source Objects

Settings

Add Component

Video Tutorial Guide for Similar: https://youtu.be/Ko_OtdNN6Os

- Example Scenes: This folder contains different types of bikes and their configurations for you to test out.

- Materials: Contains materials (all standard shaders) compatible with all the three pipelines.

- Models: Contains models of different vehicle types that the rider can ride. These models are all separated into the following moving parts:

  1. Handles
  2. Front Suspension
  3. Front Wheel
  4. Rear Wheel
  5. Rear Suspension

- Physic Materials: Contains physics materials of tyres and terrains. Inspector settings available in Moto Controller.

- Prefabs: Prefabs of the completely setup bikes. Drag and drop the vehicle types.

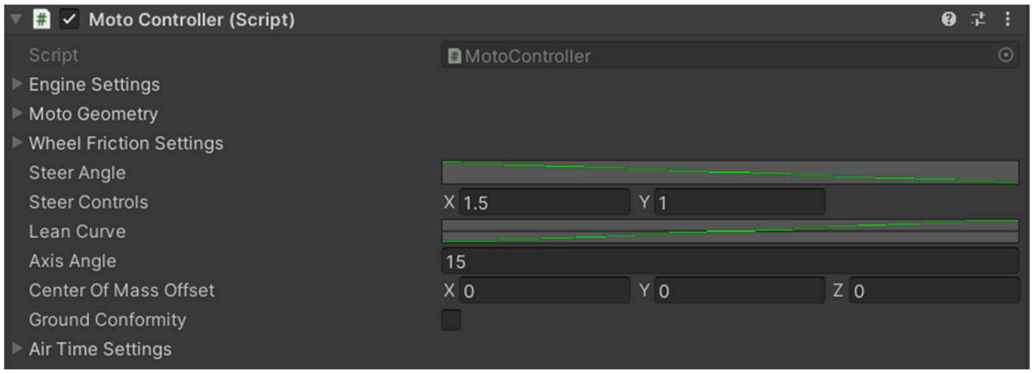
- Prototyping: Contains test environment data, textures and models.

- Scripts: Elaborated in the section below

- Textures: Contains the textures [2K - 4K] used on the materials. Provides different colors for you to choose from.

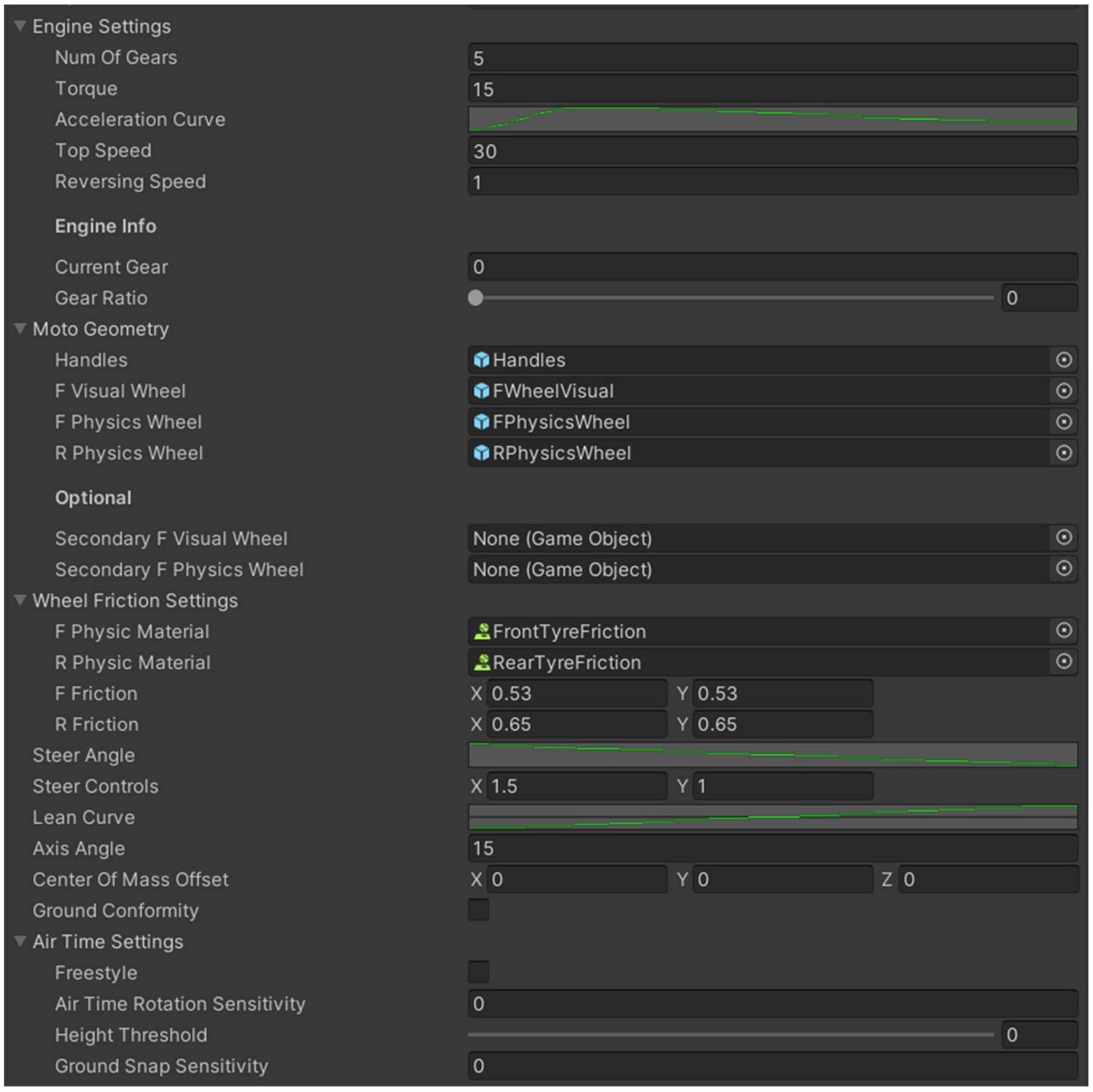
Scripts

- Moto Controller



Moto Controller (Script)	
Script	MotoController
Engine Settings	
Moto Geometry	
Wheel Friction Settings	
Steer Angle	
Steer Controls	X 1.5 Y 1
Lean Curve	
Axis Angle	15
Center Of Mass Offset	X 0 Y 0 Z 0
Ground Conformity	☐
Air Time Settings	

(EXPANDED)



Engine Settings	
Num Of Gears	5
Torque	15
Acceleration Curve	
Top Speed	30
Reversing Speed	1
Engine Info	
Current Gear	0
Gear Ratio	0
Moto Geometry	
Handles	Handles
F Visual Wheel	FWheelVisual
F Physics Wheel	FPhysicsWheel
R Physics Wheel	RPhysicsWheel
Optional	
Secondary F Visual Wheel	None (Game Object)
Secondary F Physics Wheel	None (Game Object)
Wheel Friction Settings	
F Physic Material	FrontTyreFriction
R Physic Material	RearTyreFriction
F Friction	X 0.53 Y 0.53
R Friction	X 0.65 Y 0.65
Steer Angle	
Steer Controls	X 1.5 Y 1
Lean Curve	
Axis Angle	15
Center Of Mass Offset	X 0 Y 0 Z 0
Ground Conformity	☐
Air Time Settings	
Freestyle	☐
Air Time Rotation Sensitivity	0
Height Threshold	0
Ground Snap Sensitivity	0

- (Class) Moto Geometry: The moto geometry class contains the gameObjects required to animate the motorbike and its control its movements. It accepts public members:

  1. Handles

2. Front Wheel Mesh (F Visual Wheel)
3. Front Wheel (Physics)
4. Rear Wheel (Physics)

This class works intimately with the model's anchor/pivot positions and rotates it with the help of Quaternions. Please make sure the anchors and positions of any new motorbike follow the sample motorbike.

- F Physics Wheel: This is a Unity physics joint configurable joint which has its positional motion locked on all axes and angular motion free on two axes: Rolling Forwarding + Turning. This physics body also has a Rigidbody component of a realistic mass attached to it. Uses sphere collider as it offers a great rotational curvature, better than that of a highly subdivided mesh.

- R Physics Wheel: This is a Unity physics joint configurable joint which has its positional motion locked on all axes and angular motion free on one axis: Rolling Forward. This physics body also has a Rigidbody component of a realistic mass attached to it. Uses sphere collider.

- (Class) Wheel Friction Settings: This class sets the physics materials on to the tires of the motorbike. F Friction pertains to the front tire friction and R Friction to the rear. They are of the Vector2 type. X field edits the static friction information and Y edits the dynamic friction. Please keep the values over 0.5. For more information, please read the commented scripts.

- Steer Angle: Controls the extent to which the wheel will turn from the mean position.

- Axis Angle: The amount of slant the turned wheel has. This value is purely visual. This value does not affect the physics.

- Lean Angle: This value corresponds to the visual body weight shift of the biker. Higher values contribute to a higher weight shift. This value does not affect the physics.

- Top Speed: Maximum Velocity magnitude that the motorbike is capable of reaching.

- Reversing Speed: Maximum Velocity the biker can reverse with. This does not change the reversing animation speed.

- Torque: The strength with which the rider pedals. Higher values correspond to a higher acceleration amount.

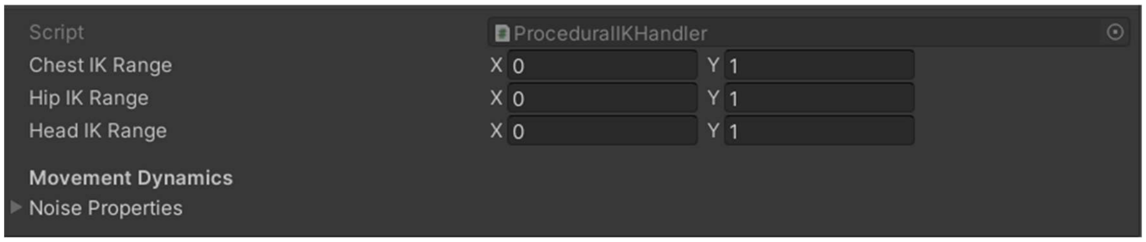
- Acceleration Curve: Acceleration booster amount. Not physically accurate. Artificially inflates speed specified in the curve.

- **COM:** Center of Mass of the vehicle.

- **Ground Conformity:** For non-gyroscopic wheel systems like the ATV, enabling ground conformity ensures that the ATV is not always upright and follows the curvature of the terrain.

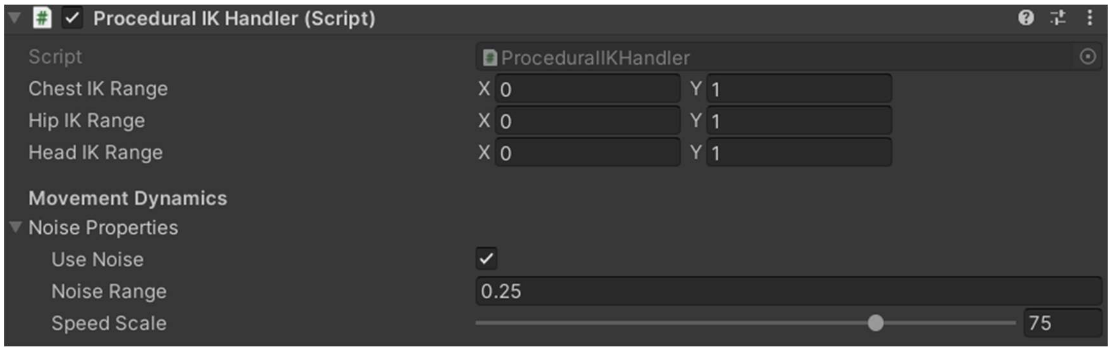
- Procedural IK Handler

  This script heavily depends on the animation rigging package and employs various IK constraints to simulate the motion of the rider.



Script	ProceduralIKHandler
Chest IK Range	X 0 Y 1
Hip IK Range	X 0 Y 1
Head IK Range	X 0 Y 1
Movement Dynamics	
▶ Noise Properties	

(EXPANDED)



Procedural IK Handler (Script)

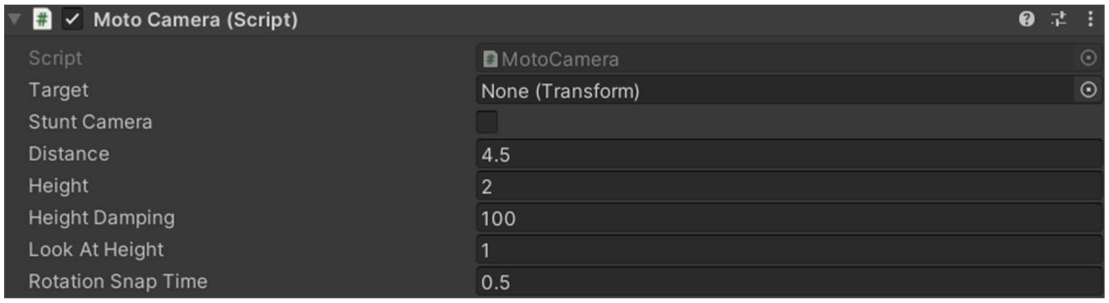
Script	ProceduralIKHandler
Chest IK Range	X 0 Y 1
Hip IK Range	X 0 Y 1
Head IK Range	X 0 Y 1
Movement Dynamics	
▼ Noise Properties	
Use Noise	✓
Noise Range	0.25
Speed Scale	75

- **Chest IK Range:** A Vector2 value which takes in the lower (X) and upper limit (Y) of the motion specified by the ChestIKTarget gameObject and modifies the weight of the movement. Weight refers to the percentage of movement between two points – very similar to Lerp.

- **Hip IK Range:** A Vector2 value which takes in the lower (X) and upper limit (Y) of the motion specified by the HipIKTarget gameObject in the prefab's hierarchy.

- **Head IK Range:** A Vector2 value which takes in the lower (X) and upper limit (Y) of the motion specified by the HeadIKTarget gameObject in the prefab's hierarchy.

- Moto Camera

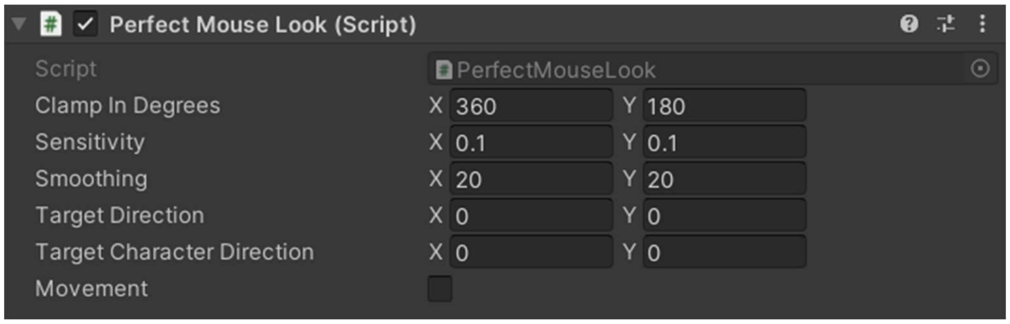


Moto Camera (Script)

Script	MotoCamera
Target	None (Transform)
Stunt Camera	☐
Distance	4.5
Height	2
Height Damping	100
Look At Height	1
Rotation Snap Time	0.5

- **Target:** GameObject to follow around.
- **Distance:** Horizontal distance from the camera
- **Height:** Vertical Distance from the camera
- **Height Damping:** Time taken for the camera to reach that height
- **Lookatheight:** Lookat offset for Height (0, Y, 0)
- **Parent Rigidbody:** Distance multiplier Calculations
- **Rotation/Distance Snap:** Time taken to reach specified rotation and distance values
- **Distance Multiplier:** Move away effect of the camera as the object speeds up

- Perfect Mouse Look

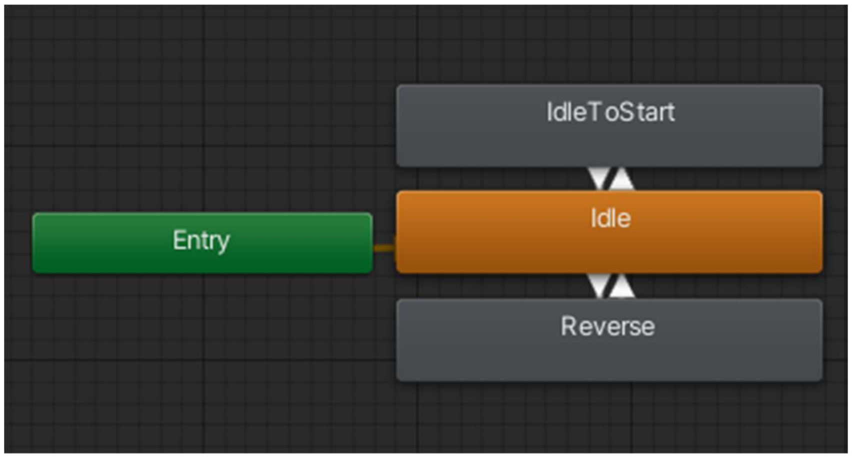


Perfect Mouse Look (Script)

Script	PerfectMouseLook			
Clamp In Degrees	X	360	Y	180
Sensitivity	X	0.1	Y	0.1
Smoothing	X	20	Y	20
Target Direction	X	0	Y	0
Target Character Direction	X	0	Y	0
Movement	☐			

When the mouse moves the perfect mouse look script is activated and the user can look around with the mouse around the target object. When the movement is stopped, the rotation snaps back to the original position smoothly

Procedural Animations



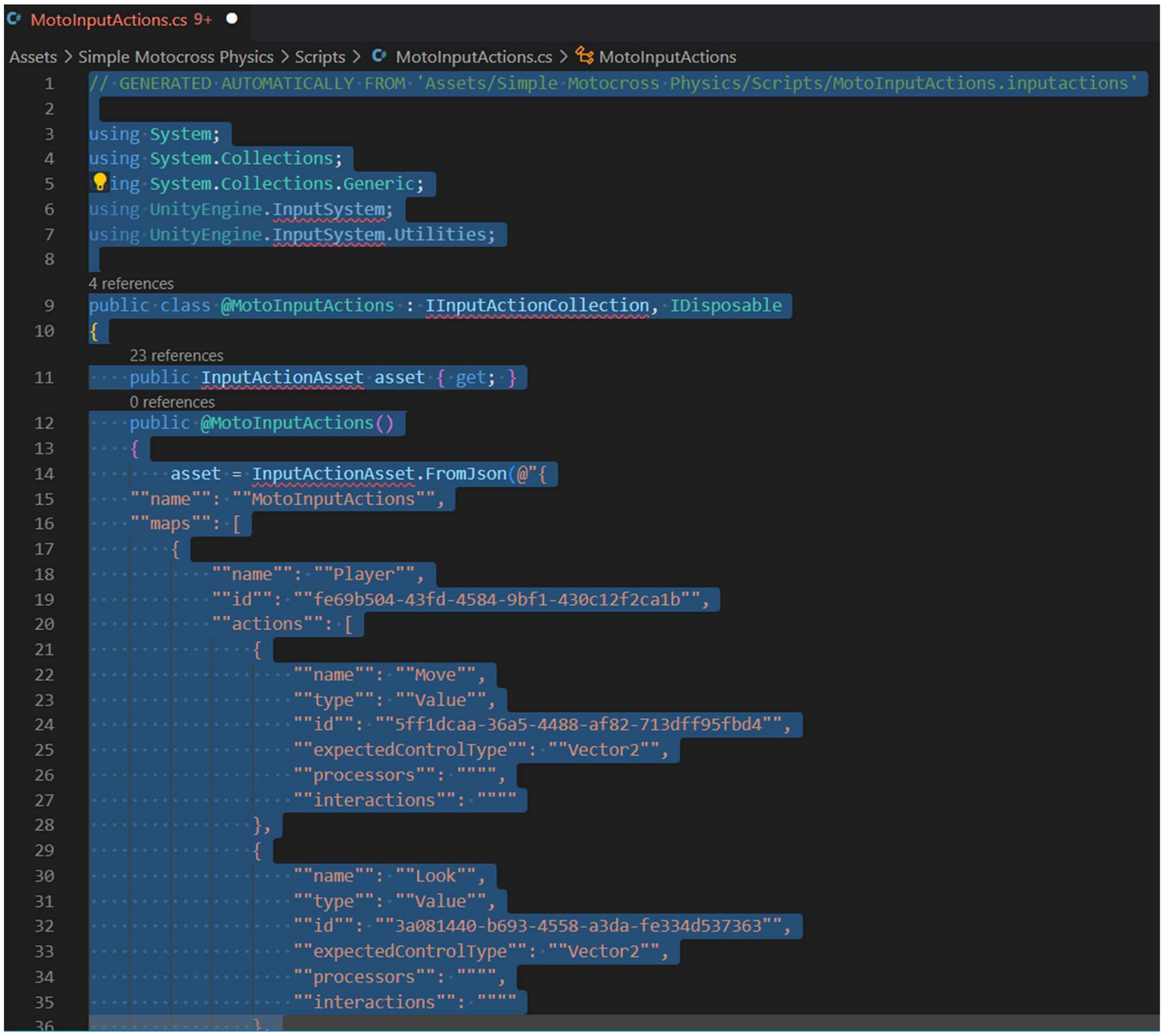
With only three animation states governing the entire movement of the biker in our asset, there are a few procedural techniques we have utilized. These are noise, and prepositioned IKs that are controlled via weight. The pseudo procedural animations are a direct outcome of the animation rigging package. All the animations are based off Unity's mechanim humanoid animation clips. These clips are directly editable in the editor itself and are highly customizable.

Using the New Input System

Simple Motocross Physics asset comes with the new Input System version 1.0.2+ compatibility so that there is a seamless transition between controls for different platforms. To move your project to the new Input system, please go to Window > Package Manager > Input System > Install.

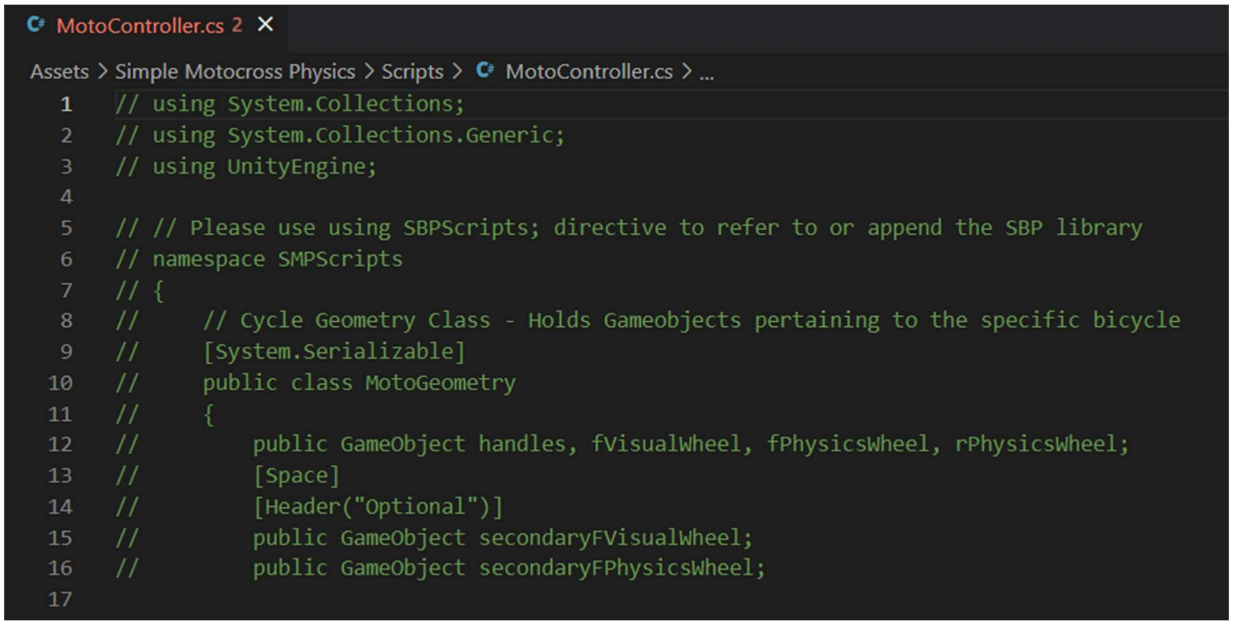
Select yes if you wish to switch to the new input system entirely. Please note that you can still use both the input systems together if you tweak the project settings.

Please uncomment the entire script named MotoInputActions.cs as shown below:
The shortcut for uncommenting is: Crtl + / or Cmd + /


MotoInputActions.cs 9+
Assets > Simple Motocross Physics > Scripts > MotoInputActions.cs > MotoInputActions
4 references
23 references
0 references

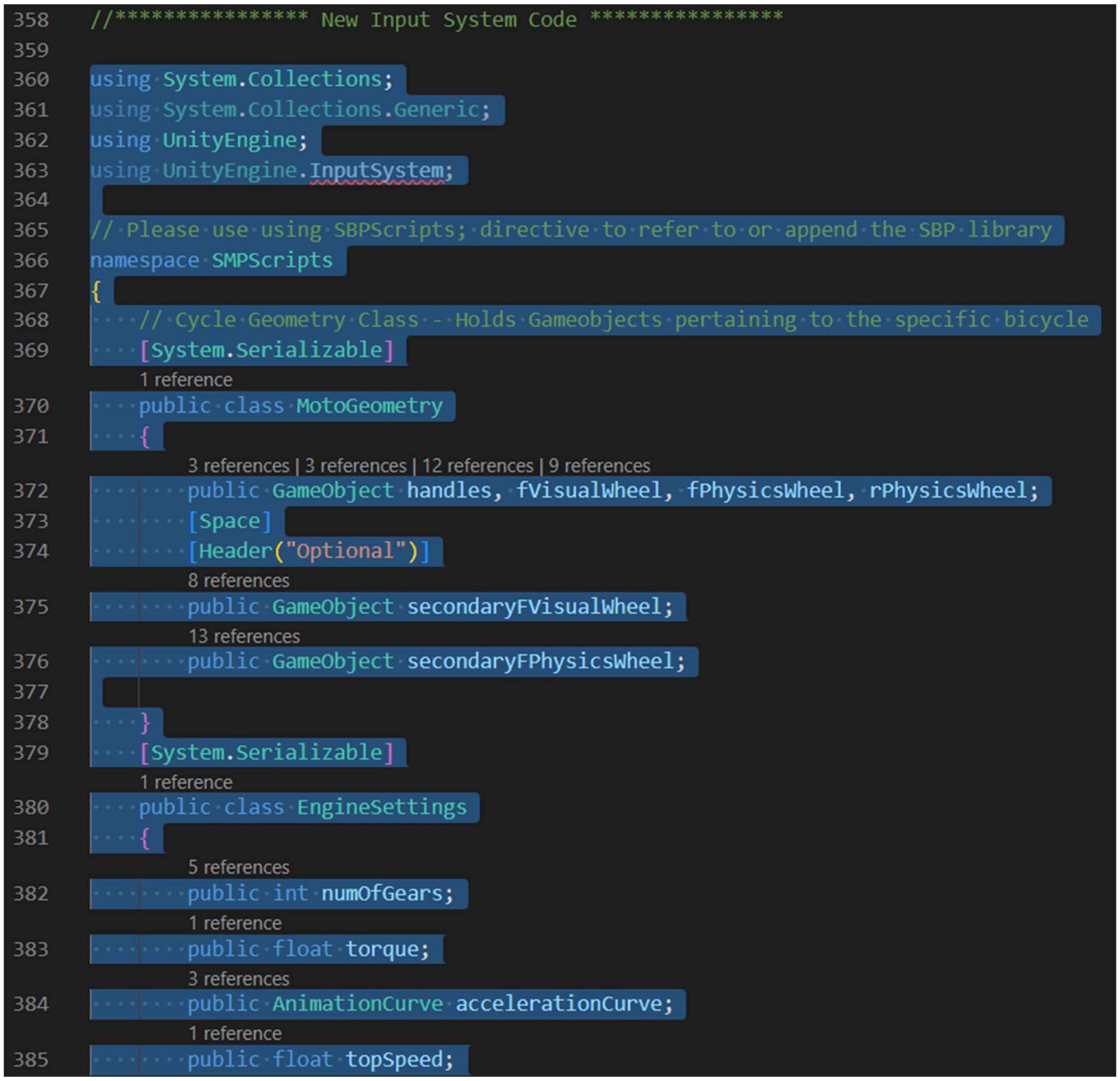
```
// GENERATED AUTOMATICALLY FROM 'Assets/Simple Motocross Physics/Scripts/MotoInputActions.inputactions'

using System;
using System.Collections;
 ing System.Collections.Generic;
using UnityEngine.InputSystem;
using UnityEngine.InputSystem.Utilities;

public class @MotoInputActions : IInputActionCollection, IDisposable
{
    public InputActionAsset asset { get; }
    public @MotoInputActions()
    {
        asset = InputActionAsset.FromJson(@"{
    ""name"": ""MotoInputActions"",
    ""maps"": [
        {
            ""name"": ""Player"",
            ""id"": ""fe69b504-43fd-4584-9bf1-430c12f2ca1b"",
            ""actions"": [
                {
                    ""name"": ""Move"",
                    ""type"": ""Value"",
                    ""id"": ""5ff1dcaa-36a5-4488-af82-713dff95fbd4"",
                    ""expectedControlType"": ""Vector2"",
                    ""processors"": """",
                    ""interactions"": """"
                },
                {
                    ""name"": ""Look"",
                    ""type"": ""Value"",
                    ""id"": ""3a081440-b693-4558-a3da-fe334d537363"",
                    ""expectedControlType"": ""Vector2"",
                    ""processors"": """",
                    ""interactions"": """"
                },
```

Go to the MotoController.cs script and comment out code till “New Input System Code”:


MotoController.cs 2
Assets > Simple Motocross Physics > Scripts > MotoController.cs > ...

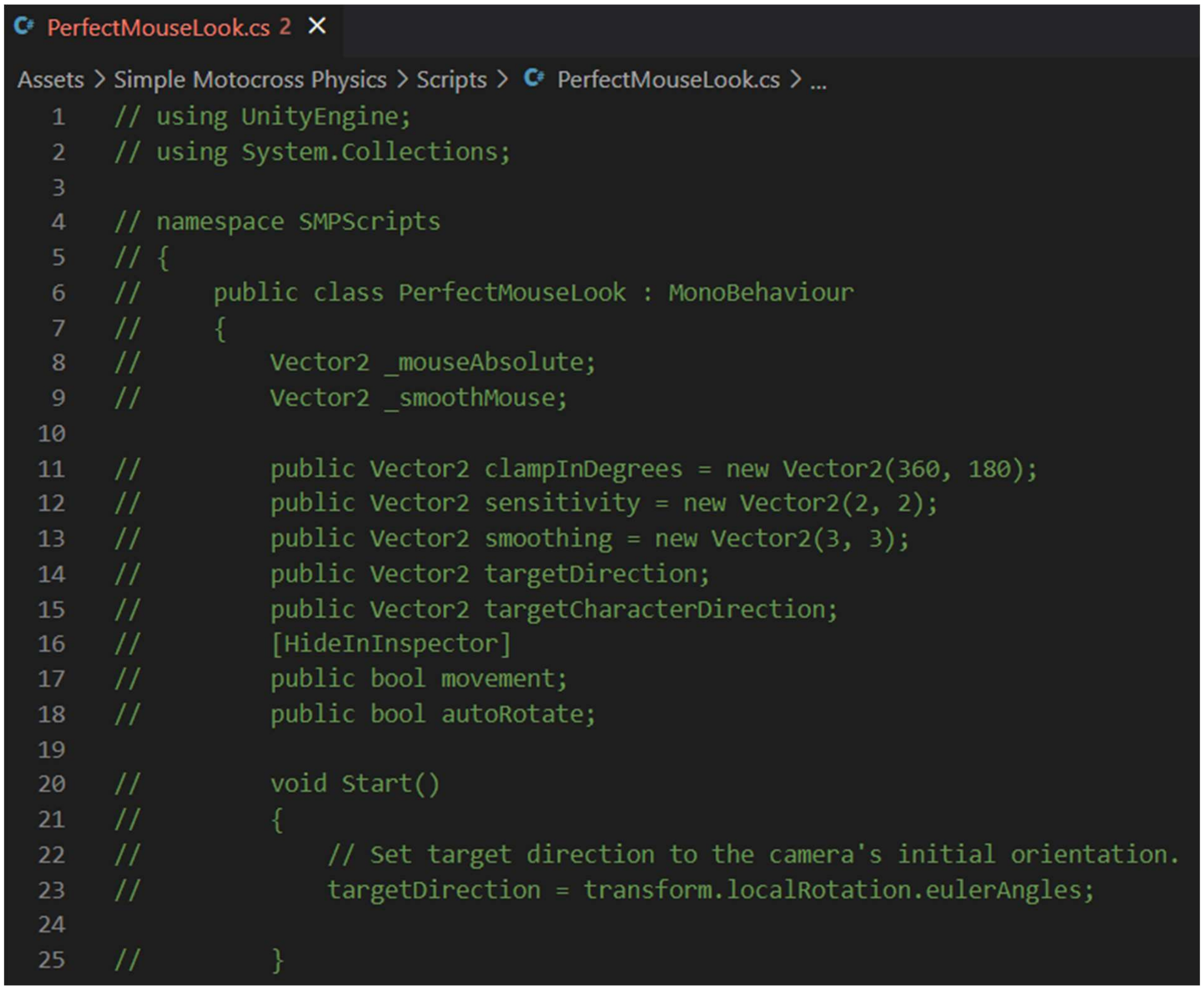
```
// using System.Collections;
// using System.Collections.Generic;
// using UnityEngine;

// // Please use using SBPScripts; directive to refer to or append the SBP library
// namespace SMPScripts
// {
//     // Cycle Geometry Class - Holds Gameobjects pertaining to the specific bicycle
//     [System.Serializable]
//     public class MotoGeometry
//     {
//         public GameObject handles, fVisualWheel, fPhysicsWheel, rPhysicsWheel;
//         [Space]
//         [Header("Optional")]
//         public GameObject secondaryFVisualWheel;
//         public GameObject secondaryFPhysicsWheel;

```

Uncomment the code from the the New Input System code till EOF (end of file pointer):



```
//**************** New Input System Code ****************

using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.InputSystem;

// Please use using SBPScripts; directive to refer to or append the SBP library
namespace SMPScripts
{
    // Cycle Geometry Class - Holds Gameobjects pertaining to the specific bicycle
    [System.Serializable]
    public class MotoGeometry
    {
        public GameObject handles, fVisualWheel, fPhysicsWheel, rPhysicsWheel;
        [Space]
        [Header("Optional")]
        public GameObject secondaryFVisualWheel;
        public GameObject secondaryFPhysicsWheel;

    }
    [System.Serializable]
    public class EngineSettings
    {
        public int numOfGears;
        public float torque;
        public AnimationCurve accelerationCurve;
        public float topSpeed;
```

Repeat the aforementioned process for the PerfectMouseLook.cs script, comment out code till “New Input System Code”:

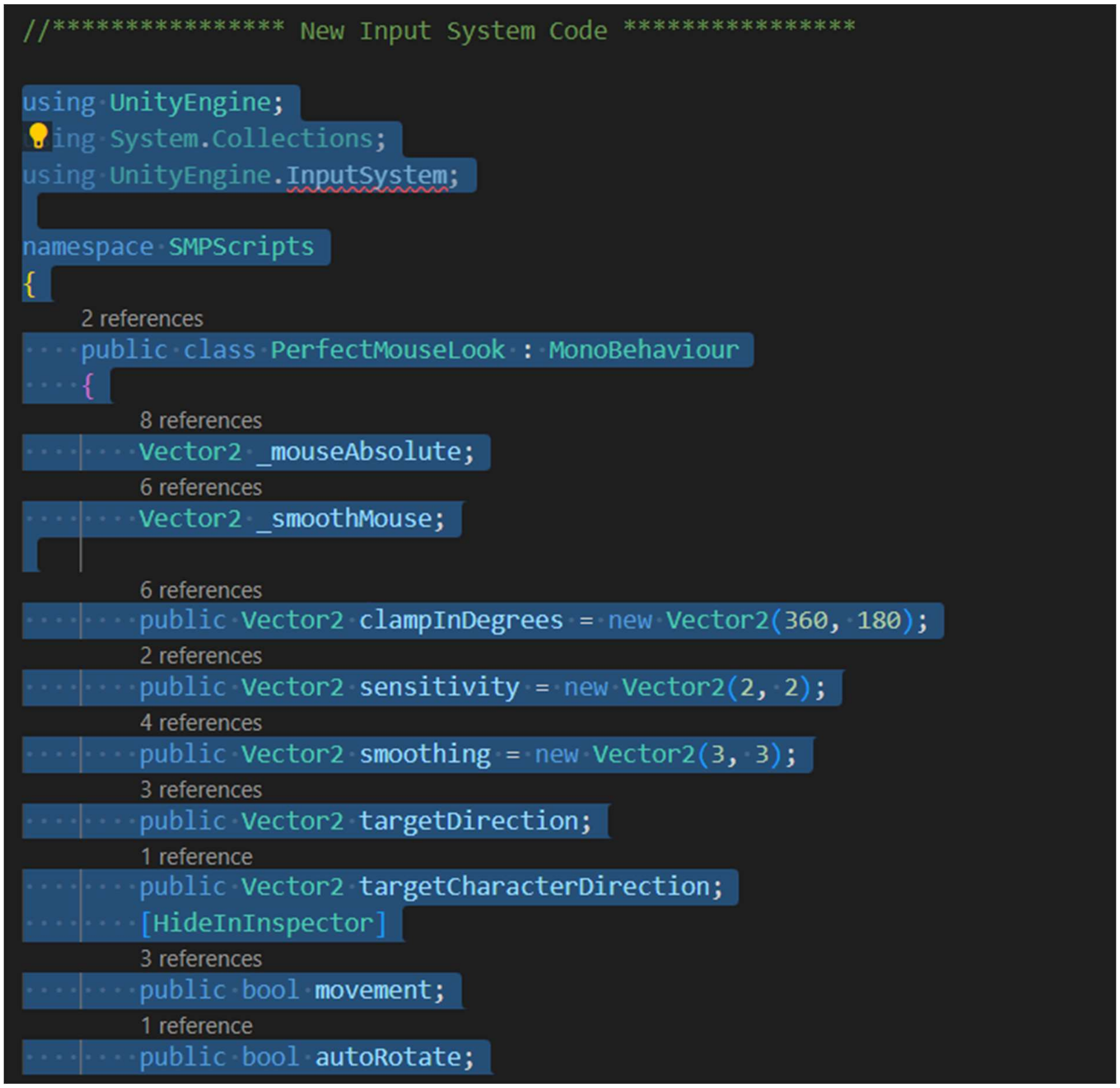


PerfectMouseLook.cs 2

Assets > Simple Motocross Physics > Scripts > PerfectMouseLook.cs > ...

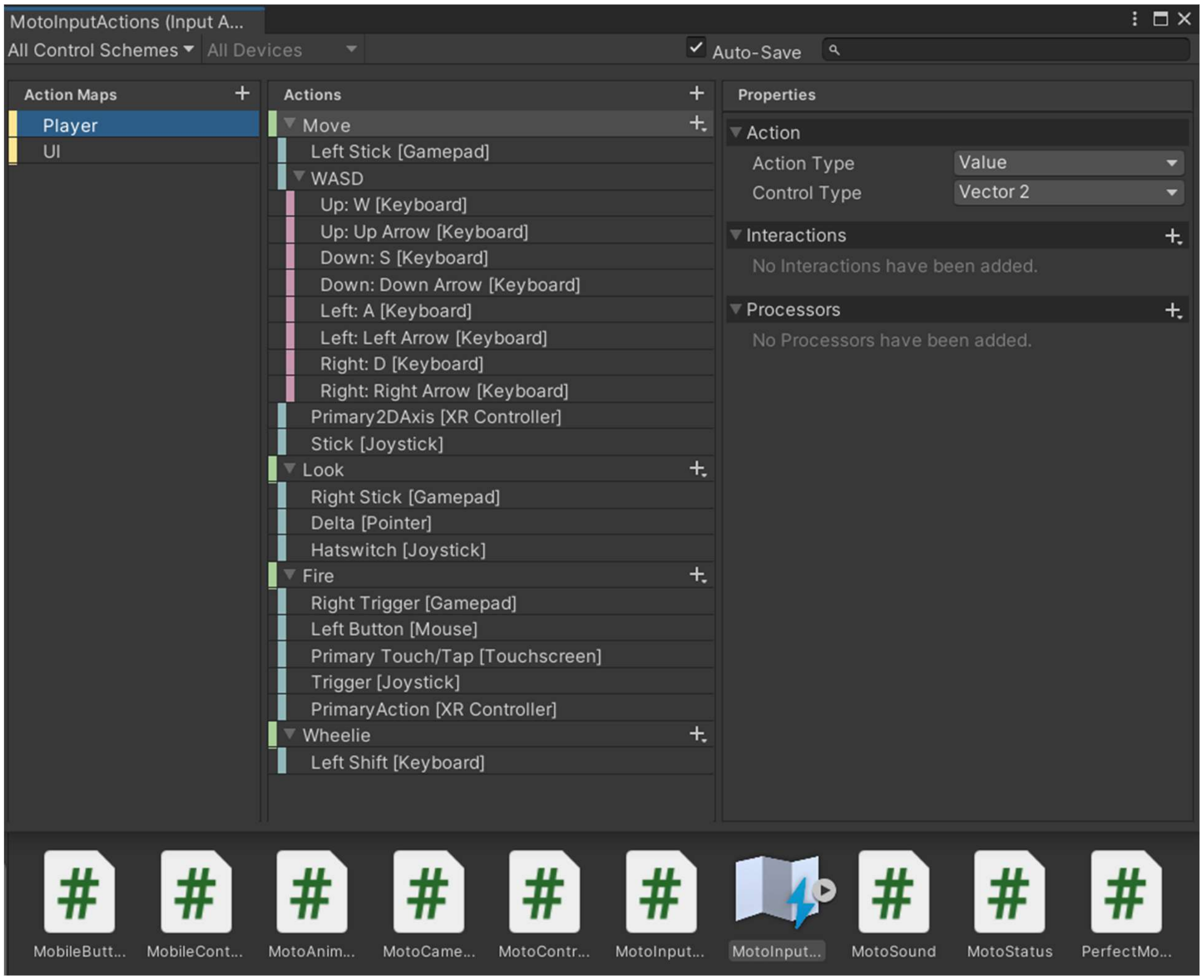
```
// using UnityEngine;
// using System.Collections;

// namespace SMPScripts
// {
//     public class PerfectMouseLook : MonoBehaviour
//     {
//         Vector2 _mouseAbsolute;
//         Vector2 _smoothMouse;

//         public Vector2 clampInDegrees = new Vector2(360, 180);
//         public Vector2 sensitivity = new Vector2(2, 2);
//         public Vector2 smoothing = new Vector2(3, 3);
//         public Vector2 targetDirection;
//         public Vector2 targetCharacterDirection;
//         [HideInInspector]
//         public bool movement;
//         public bool autoRotate;

//         void Start()
//         {
//             // Set target direction to the camera's initial orientation.
//             targetDirection = transform.localRotation.eulerAngles;

//         }
```

Uncomment the code from the New Input System code till EOF (end of file pointer):



```
//**************** New Input System Code ****************

using UnityEngine;
using System.Collections;
using UnityEngine.InputSystem;

namespace SMPScripts
{
    2 references
    public class PerfectMouseLook : MonoBehaviour
    {
        8 references
        Vector2 _mouseAbsolute;
        6 references
        Vector2 _smoothMouse;

        6 references
        public Vector2 clampInDegrees = new Vector2(360, 180);
        2 references
        public Vector2 sensitivity = new Vector2(2, 2);
        4 references
        public Vector2 smoothing = new Vector2(3, 3);
        3 references
        public Vector2 targetDirection;
        1 reference
        public Vector2 targetCharacterDirection;
        [HideInInspector]
        3 references
        public bool movement;
        1 reference
        public bool autoRotate;
```

You can configure the key binding and controls from the Input Manager GUI from the Scripts folder:



Action Maps
- Player
- UI

Actions
- Move
  - Left Stick [Gamepad]
  - WASD
    - Up: W [Keyboard]
    - Up: Up Arrow [Keyboard]
    - Down: S [Keyboard]
    - Down: Down Arrow [Keyboard]
    - Left: A [Keyboard]
    - Left: Left Arrow [Keyboard]
    - Right: D [Keyboard]
    - Right: Right Arrow [Keyboard]
  - Primary2DAxis [XR Controller]
  - Stick [Joystick]
- Look
  - Right Stick [Gamepad]
  - Delta [Pointer]
  - Hatswitch [Joystick]
- Fire
  - Right Trigger [Gamepad]
  - Left Button [Mouse]
  - Primary Touch/Tap [Touchscreen]
  - Trigger [Joystick]
  - PrimaryAction [XR Controller]
- Wheelie
  - Left Shift [Keyboard]

Properties

Action

Action Type	Value
Control Type	Vector 2

Interactions

No Interactions have been added.

Processors

No Processors have been added.

Documentation / Guides

There are numerous guides in the scenes folder that will help you develop your game in the direction of a feature that you desire your game to have.
We have mobile setup instructions, ragdoll guides and stunt mode setup guides available in the package that you can follow if you want these features for your game.

If you have any questions, please feel free to contact us at info@aikodex.com.

AiKodex