FORCE FEEDBACK TOOLKIT

INTRODUCTION

This toolkit provides a set of functions for using Force Feedback and Direct Input within Unity3D. This toolkit has been specifically designed to work in Unity Free.

This toolkit is a developer's toolkit. At least basic knowledge of programming will be required to use this toolkit.

Note: You must copy the FFWheelInput.DLL into the root directory of your project. i.e. The parent directory of your assets folder.

PACKAGE CONTENTS

The package by default exists under the Standard Assets folder. This is to ensure the scripts are compiled before other scripts in your project. This will JavaScript scripts can use FFB.

FORCE FEEDBACK TOOLKIT FILES

1. **FFWheelInput.DLL:** Direct input controller. Manages the direct input device and sends force feedback requests to the game controller. This object must exist in the root directory of the project. i.e. in the parent directory of the Assets folder.
2. **DInputProxy.cs:** This acts as a proxy between unity and the FFWheelInput.DLL.
   RawJoystickState.cs: Class for marshalling game controller state.
3. **ControllerProperites.**cs: Class for marshalling controller properties.

SAMPLE SCENE PROGRAMS

4. **WheelInput.cs:** Takes raw input from DinputProxy.cs and processes it for easier use by the application. This class is a singleton, i.e. only one copy of it should exist at any time.
5. **InputScan.cs:** This class scans for direct input activity. It is used to map input controls to actions in WheelMenu.
6. **WheelMenu.cs:** This class is typically attached to the main camera in the scene. It should be the only component to reference to WheelInput. It also displays direct input setup menu when "M" is pressed.
7. **FFB Scene:** Example scene that simply displays the WheelMenu GUI Menu to test force feedback effects and map game controller input. Shows game controller input values for mapped actions.
8. **FFBGenerator.cs:** Will generate force feedback instructions to game controller when attached and configured on a vehicle that is using Wheel Colliders. Used in MultiController sample game and also for integration with Edy's vehicle physics.

MULTIPLECONTROLLERS FOLDER

9. **Ball.prefab & car.prefab:** Prefabs for game.
10. **FFBCarController.cs:** Controls movement of car prefab.
11. **FFBGame:** Game controller. Keeps score, instantiates prefabs.
12. **MultipleControllers Scene:** Sample game that demonstrates force feedback, multiple controllers and direct input.

POPULAR VEHICLE PACKAGE EXAMPLES

13. **UnityCar.ZIP:** This file contains files for the Unity Car Example.
14. **Edy.Zip:** This file contains files for the Edy's Vehicle Physics Example
15. **CarTutorial.Zip:** This file contains files for the Car Tutorial Example.

DirectX 9

Please note, this package requires DirectX 9. This can be downloaded from
http://www.microsoft.com/en-us/download/details.aspx?id=8109

GETTING STARTED

CAR PACKAGE EXAMPLES

If you have Unity Car, Edy's Vehicle Physics or Car Tutorial, I suggest you start with the specific examples that have been provided for those packages. Instructions can be found towards the end of this document.

SAMPLE SCENE

Before you start integrating Force Feedback with your own package, I suggest you take a look at the MultipleControllers sample scene. This scene is very simple and demonstrates how to use multiple controllers, direct input and force feedback.

YOUR OWN GAME

If you are not using one of the car packages mentioned above then the following steps should help you get started integrating force feedback into your game.

1. Load the package into your project ☺
2. Copy FFWheelInput.DLL to the root directory of your project. That is the parent directory of the Assets folder.
3. Attach the WheelMenu.cs component to your main camera. This component has been designed for Cars, but you can change the code to suit your game. It provides functions for Direct Input and Force Feedback.
   In the inspector ensure that ***"Enable Menu"*** is checked, this makes the GUI menu available when the "M" key is pressed. Also ensure that ***"Show FFB Test"*** is checked, this displays the force feedback test controls on the GUI menu when it is displayed.
   "Show at Startup" and ***"Show Raw Input"*** should be **off**. These make the menu available when the game first starts and show raw data from the joystick rather than the mapping menu, respectively.
4. You should now run your scene. Press the "M" key to see the Force Feedback menu. If you have a controller attached, it's name should be displayed below the "Controller Setup" button. If you have more than one controller attached, you can cycle through them by pressing the "Prev" and "Next" buttons.
   Try mapping one of the functions with your controller, for example, press the set button next to "Steering Axis", then move one of the axis on your controller. This axis should now be mapped to this function, when you again move the axis on the controller the input value should be updated.
5. If your controller has force feedback, you can use the options at the bottom of the menu to send test force feedback instructions to the controller.
6. Using WheelMenu is the easiest way to access the toolkit functions. You can now create a reference to it in your own scripts. To do this in C# use the following command:

```
WheelMenu wheelMenu = Camera.main.GetComponent<WheelMenu>();
```

7. To access the toolkit functions use Dinput property of WheelMenu, eg:

```
wheelMenu.dinput.PlayBumpyRoadEffect(50);
```

8. A full list of functions available in the toolkit are defined in the next section.

FORCE FEEDBACK FUNCTIONS

Force Feedback and Direct Input functions are available through the WheelMenu script and it's Dinput property, i.e. WheelMenu.dinput. Some of the functions have been derived using the Logitech SDK. The descriptions for these functions have been copied from the Logitech SDK documentation.

Up to four game controllers can be accessed at once through the toolkit. Many of the functions have been overloaded to accept a specific game controller number or to use the default controller.

DIRECT INPUT AND FORCE FEEDBACK

The following methods affect both Direct Input and Force feedback methods

Void PrevController()

Set the default controller to the next available controller. If only one controller is available, then will not change current controller.

Void NextController()

Set the controller to the next available controller. If only one controller is available, then will not change current controller.

DIRECT INPUT METHODS

The following methods are for Direct Input.

Void GetInput() or GetInput(int Index)

Update the input buffer with the current controller values.

Can be called with INDEX parameter to specify a specific controller, otherwise will use default controller.

Void SetCombined(int value) or SetCombined(int index, int value)

Set the accelerator and brake to use the same axis on the controller. An example of how this can be used is when you are using a Joystick or Joypad and want the forward movement of the stick to be acceleration and the backward movement to be brake.

The value parameter should be "1" for setting combined on and "0" for off.

Can be called with INDEX parameter to specify a specific controller, otherwise will use default controller.

Bool Combined()

Returns true if combined is on for accelerator and brake.

Float GetSteer(int sensitivity) or GetSteer(int index, float sensitivity)

Returns the current input for steering.

The sensitivity parameter reduces the scale of the returned value. This is useful if you wish to make the steering wheel less responsive as the vehicle moves faster.

Can be called with INDEX parameter to specify a specific controller, otherwise will use default controller.

Float GetThrottle() or GetThrottle(int index)

Returns the current input for the throttle.

Can be called with INDEX parameter to specify a specific controller, otherwise will use default controller.

Float GetBrake() or GetBrake(int index)

Returns the current input for the brake.

Can be called with INDEX parameter to specify a specific controller, otherwise will use default controller.

Float GetClutch() or GetClutch(int index)

Returns the current input for the clutch.

Can be called with INDEX parameter to specify a specific controller, otherwise will use default controller.

Float GetHandbrake() or GetHandbrake(int index)

Returns the current input for the handbrake.

Can be called with INDEX parameter to specify a specific controller, otherwise will use default controller.

Bool GetStartEngine() or GetStartEngine(int index)

Returns 1 if start engine button is pressed otherwise 0.

Can be called with INDEX parameter to specify a specific controller, otherwise will use default controller.

ProcessAnalogueInput(int index, int inputId, float sensitivity)

Returns current analogue input from game controller.

Index – Is the game controller number.

InputId – is the Axis ID number. These are defined as follows:

```
switch (inputId)
{
    case 100: retVal = "XAxis"; break;
    case 101: retVal = "YAxis"; break;
    case 102: retVal = "ZAxis"; break;
    case 103: retVal = "XRot"; break;
    case 104: retVal = "YRot"; break;
    case 105: retVal = "ZRot"; break;
    case 106: retVal = "Slider0"; break;
    case 107: retVal = "Slider1"; break;
    case 108: retVal = "POV0"; break;
    case 109: retVal = "POV1"; break;
    case 110: retVal = "POV2"; break;
    case 111: retVal = "POV3"; break;
    case 112: retVal = "ForceSlider0"; break;
    case 113: retVal = "ForceSlider1"; break;
    case 114: retVal = "ForceX"; break;
    case 115: retVal = "ForceY"; break;
    case 116: retVal = "ForceZ"; break;
    case 117: retVal = "TorqueX"; break;
    case 118: retVal = "TorqueY"; break;
    case 119: retVal = "TorqueZ"; break;
    case 120: retVal = "VelS0"; break;
    case 121: retVal = "VelS1"; break;
```

```
            case 122: retVal = "VelX"; break;
            case 123: retVal = "VelY"; break;
            case 124: retVal = "VelZ"; break;
        }
```

Sensitivity – Factor that the returned value is multiplied by.

Bool ShiftUp() or ShiftUp(int index)

Returns true if the up shift button is pressed.

Can be called with INDEX parameter to specify a specific controller, otherwise will use default controller.

Bool ShiftDown() or ShiftDown(int index)

Returns true if the down shift button is pressed.

Can be called with INDEX parameter to specify a specific controller, otherwise will use default controller.

Int GetManualGear() or GetManualGear(int index)

Returns the current manual gear on the H-Shifter.

Can be called with INDEX parameter to specify a specific controller, otherwise will use default controller.

Float getInputStatus(GameOptions.InputType iType) or getInputStatus(int index, GameOptions.InputType iType)

Returns the current input value for the mapped input axis/button.

Type can be:

```
public class GameOptions
{
    public enum InputType { Throttle, Brake, Steer, Clutch, Handbrake, ShiftUp,
ShiftDown, StartEngine, FirstGear, SecondGear, ThirdGear, FourthGear, FifthGear,
SixthGear, ReverseGear, Combined, None };

}
```

Can be called with INDEX parameter to specify a specific controller, otherwise will use default controller

Float getInputStatus(int inputId) or getInputStatus(int index, int inputId)

Returns the current input value for the axis/button on the controller. An inputId of 0 to 40 wheel return the corresponding button. A value of 100 to 124 will return the analogue axis:

```
switch (inputId)
{
    case 100: retVal = "XAxis"; break;
    case 101: retVal = "YAxis"; break;
    case 102: retVal = "ZAxis"; break;
    case 103: retVal = "XRot"; break;
    case 104: retVal = "YRot"; break;
    case 105: retVal = "ZRot"; break;
    case 106: retVal = "Slider0"; break;
    case 107: retVal = "Slider1"; break;
    case 108: retVal = "POV0"; break;
    case 109: retVal = "POV1"; break;
    case 110: retVal = "POV2"; break;
    case 111: retVal = "POV3"; break;
    case 112: retVal = "ForceSlider0"; break;
    case 113: retVal = "ForceSlider1"; break;
    case 114: retVal = "ForceX"; break;
    case 115: retVal = "ForceY"; break;
    case 116: retVal = "ForceZ"; break;
    case 117: retVal = "TorqueX"; break;
    case 118: retVal = "TorqueY"; break;
    case 119: retVal = "TorqueZ"; break;
    case 120: retVal = "VelS0"; break;
    case 121: retVal = "VelS1"; break;
    case 122: retVal = "VelX"; break;
    case 123: retVal = "VelY"; break;
    case 124: retVal = "VelZ"; break;
}
```

String getname(GameOptions.InputType iType)

Returns the name of the mapped input axis/button to this input for the default controller.

String getName(int inputId)

Returns the name of the axis/button. 0 to 40 are buttons and 100 to 124 are analogue axis.

Void LoadInputSettings(int deviceId, string devicePID)

Loads the saved mappings for the device. See SaveInputSettings for further details.

DeviceID is the game controller number.

DevicePID is the product ID for the game controller.

Void SaveInputSettings() or SaveInputSettings(int deviceId, String devicePID)

Save the current mappings for the device. These are saved in a file in the root folder of the game. The file is named "controllerMap_12345.dta" where "12345" is the product ID as returned in Direct Input or passed as a parameter to the function.

The controller mappings can be loaded next time the game is run using the LoadInputSettings method.

Void ClearInput(GameOptions.InputType t)
or ClearInput(int deviceId, GameOptions.InputType t)

This will clear the mapped axis/button for the specified type.

FORCE FEEDBACK METHODS

The following methods are for sending force feedback instructions to the game controller.

Bool IsConnected() or IsConnected(int index)

Returns true if the game controller is connected.

Can be called with INDEX parameter to specify a specific controller, otherwise will use default controller.

Bool HasForceFeedback() or HasForceFeedback(int index)

Returns true if the game controller supports force feedback.

Can be called with INDEX parameter to specify a specific controller, otherwise will use default controller.

Bool IsGamepad() or IsGamePad(int index)

Returns true if the game controller is a gamepad.

Can be called with INDEX parameter to specify a specific controller, otherwise will use default controller.

String GetControllerName() or GetControllerName(int index)

Returns the friendly product name for the game controller.

Can be called with INDEX parameter to specify a specific controller, otherwise will use default controller.

Void PlayRumble(int leftMotor, int rightMotor) or PlayRumble(int index, int leftMotor, int rightMotor)

Plays rumble effects on gamepads. To turn of rumble, call PlayRumble with left & right motor values of 0.

leftMotor – Valid values are 0 to 100,000.

rightMotor – Valid values are 0 to 100,000.

Can be called with INDEX parameter to specify a specific controller, otherwise will use default controller.

Void PlayBumpyRoadEffect(int x) or PlayBumptyRoadEffect(int index, int x)

Play a surface effect that feels like driving on a bumpy road (like on cobblestones for example).

X - Specifies the magnitude of the bumpy road effect. Valid ranges for X are 0 to 100. Values higher than 100 are silently clamped.

Can be called with INDEX parameter to specify a specific controller, otherwise will use default controller.

StopBumpyRoadEffect() or StopBumpyRoadEffect(int index)

Stop the bumpy road effect.

Can be called with INDEX parameter to specify a specific controller, otherwise will use default controller.

PlaySlipperyRoadEffect(int x) or PlaySlipperyRoadEffect(int index, int x)

Play a slippery road effect (snow, ice).

X - Specifies the magnitude of the slippery road effect. Valid ranges for X are 0 to 100. 100 corresponds to the most slippery effect.

Can be called with INDEX parameter to specify a specific controller, otherwise will use default controller.

StopSlipperyRoadEffect() or StopSlipperyRoadEffect(int index)

Stop the slippery road effect.

Can be called with INDEX parameter to specify a specific controller, otherwise will use default controller.

PlayDamperForce(int x) or PlayDamperForce(int index, int x)

Play the damper force.

x - specify the slope of the effect strength increase relative to the amount of deflection from the center of the condition. Higher values mean that the saturation level is reached sooner. Valid ranges are -100 to 100. Any value outside the valid range is silently clamped. -100 simulates a

very slippery effect, +100 makes the wheel/joystick very hard to move, simulating the car at a stop or in mud.

Can be called with INDEX parameter to specify a specific controller, otherwise will use default controller.

StopDamperForce() or StopDamperForce(int index)

stop the damper force.

Can be called with INDEX parameter to specify a specific controller, otherwise will use default controller.

PlayDirtRoadEffect(int x) or PlayDirtRoadEffect(int index, int x)

Play a surface effect that feels like driving on a dirt road.

x - Specifies the magnitude of the dirt road effect. Valid ranges for x are 0 to 100. Values higher than 100 are silently clamped.

Can be called with INDEX parameter to specify a specific controller, otherwise will use default controller.

StopDirtRoadEffect() or StopDirtRoadEffect(int index)

stop the dirt road effect.

Can be called with INDEX parameter to specify a specific controller, otherwise will use default controller.

PlaySpringForce(Int32 offsetPercentage, Int32 saturationPercentage, Int32 coefficientPercentage) or PlaySpringForce(int index, Int32 offsetPercentage, Int32 saturationPercentage, Int32 coefficientPercentage)

Play the spring force.

offsetPercentage - valid range is -100 to 100. Specifying 0 centers the spring. Any values outside this range are silently clamped.

saturationPercentage - Specify the level of saturation of the spring force effect. The saturation stays constant after a certain deflection from the center of the spring. It is comparable to a magnitude. Valid ranges are 0 to 100. Any value higher than 100 is silently clamped.

coefficientPercentage - Specify the slope of the effect strength increase relative to the amount of deflection from the center of the condition. Higher values mean that the saturation level is reached sooner. Valid ranges are -100 to 100. Any value outside the valid range is silently clamped.

Can be called with INDEX parameter to specify a specific controller, otherwise will use default controller.

StopSpringForce() or StopSpringForce(int index)

Stop the spring force.

Can be called with INDEX parameter to specify a specific controller, otherwise will use default controller.

PlayAirbourne() or PlayAirbourne(int index)

play an effect that simulates a car that is airborne or where the front wheels do not touch the ground.

Can be called with INDEX parameter to specify a specific controller, otherwise will use default controller.

StopAirbourne() or StopAirbourne(int index)

stop the car airborne effect and resume any forces that were playing before the car was airborne.

Can be called with INDEX parameter to specify a specific controller, otherwise will use default controller.

PlayFrontalCollisionForce(int x) or PlayFrontalCollisionForce(int index, int x)

Play a frontal collision force.

x - specifies the magnitude of the frontal collision force effect. Valid ranges for x are 0 to 100. Values higher than 100 are silently clamped.

Can be called with INDEX parameter to specify a specific controller, otherwise will use default controller.

PlaySideCollisionForce(int x) or PlaySideCollisionForce(index x, int x)

play a side collision force.

x - Specifies the magnitude of the side collision force effect. A negative value reverses the direction of the force. Valid ranges for x are -100 to 100. Any values outside the valid range are silently clamped.

Can be called with INDEX parameter to specify a specific controller, otherwise will use default controller.

SetLeds(float value, float min, float max)
or SetLeds(int index, float value, float min, float max)

play LEDs on G27.

value - current RPM.
min - RPM when first LEDs are to turn on.
max - just below this RPM, all LEDs will be on. Just above, all LEDs will start flashing.

Can be called with INDEX parameter to specify a specific controller, otherwise will use default controller.

PlayConstantForce(int forceDirection)
or PlayConstantForce(int index, int forceDirection)

A constant force works best when continuously updated with a value tied to the physics engine. Tie the steering wheel/joystick to the car's physics engine via a vector force. This will create a centering spring effect, a sliding effect, a feeling for the car's inertia, and depending on the physics engine it should also give side collisions (wheel/joystick jerks in the opposite way of the wall the car just touched).

The vector force could for example be calculated from the lateral force measured at the front tires. This vector force should be 0 when at a stop or driving straight. When driving through a turn or when driving on a banked surface the vector force should have a magnitude that grows in a proportional way.

`forceDirection` - Specifies the magnitude of the constant force effect. A negative value reverses the direction of the force. Valid ranges for `forceDirection` are -100 to 100. Any values outside the valid range are silently clamped.

Can be called with INDEX parameter to specify a specific controller, otherwise will use default controller.

SetForce2DCoordinate(float x, float y)
or SetForce2DCoordinate(int index, float x, float y)

Set force direction on a joystick with 2 force feedback axis.

x – specifies the horizontal force. Specifies the magnitude of the constant force effect. A negative value reverses the direction of the force. Valid ranges for x are -100 to 100. Any values outside the valid range are silently clamped

y – specifies the vertical force. Specifies the magnitude of the constant force effect. A negative value reverses the direction of the force. Valid ranges for y are -100 to 100. Any values outside the valid range are silently clamped

Can be called with INDEX parameter to specify a specific controller, otherwise will use default controller.

EDY'S VEHICLE PHYSICS EXAMPLE

This example can be used to modify a clean install of the ***Edy's Vehicle Physics*** package available from the Unity Asset Store. After making these modifications the aligning forces on the front wheels will be played on your force feedback controller.

INSTRUCTIONS FOR ADDING FORCE FEEDBACK

Step 1: Import the FFB Controller package into the Edy's Vehicle Physics project

Step 2: Copy FFWheelInput.dll from FFB Folder to root folder of project (i.e. parent folder of Assets)

Step 3: Put the WheelMenu component onto the main camera.

Step 4: Put the FFBGenerator component onto the vehicle you wish to test. Adjust the FZ Factor in the inspector to 0.0001. You can adjust FFB Scale and Fz Factor to fine tune force feedback strength.

Step 5: The vehicles front Wheelcolliders WheelFL and WheelFR must be dragged onto the Wheel Front Left and Wheel Front Right properties of FFBGenerator in the inspector.

Step 6: Run the scene, force feedback should be playing through your game controller.

INSTRUCTIONS FOR USING DIRECT INPUT

Step 1: **Steps above must be done first.**

Step 2: The FFB Controller Package includes a file ***edy.zip***. This contains a single file, ***CarExternalDInput.js***, unzip this file into your project.

Step 4: Add the component ***CarExternalDInput.js*** onto the vehicle you are testing.

Step 5: Ensure "External Input" is checked on in the ***CarSettings*** component for the vehicle.

Step 7: Now play the scene. Press the "M" key, this will display the FFB menu. Map your controller to the appropriate functions. Save and close the FFB menu. Your vehicles should now be using direct input.

CAR TUTORIAL EXAMPLE

This example can be used to modify a clean install of the ***Car Tutorial*** package available from the Unity Asset Store. After making these modifications the aligning forces on the front wheels will be played on your force feedback device. This example only works with the Alternate Physics Model.

Instructions:

Step 1: Import the FFB Controller package into the Car Tutorial project

Step 2: Copy FFWheelInput.dll from FFB Folder to root folder of project (i.e. parent folder of Assets)

Step 3: A minor change is required to Wheel.cs. The normalForce & localVelo variables need to be made public. That is change:

```
float normalForce;   ➔ public float normalForce;
Vector3 localVelo; ➔ public Vector3 localVelo;
```

Step 3: The FFB Controller Package includes a file ***CarTutorial.zip***. This contains a single file, ***CarController.cs.*** Override the file with this name in \Assets\~AlternatePysicsModel\CarController.cs. Please ensure you back up the original file before unzipping the file to this directory.
After the file has been copied, ensure that the Arcade Lambo still has the component CarController defined and the link hasn't been lost.

Step 4: Add the ***WheelMenu.cs*** component to the ***MAIN_CAMERA*** object

Step 5: Now play the scene. Press the "M" key, this will display the FFB menu. Map your controller to the appropriate functions. Save and close the FFB menu. Your vehicles should now have Force Feedback enabled.

UNITY CAR EXAMPLE

Note: This example only works with a single vehicle in the scene and only for the first controller attached to your system.

INSTRUCTIONS FOR ADDING FORCE FEEDBACK

Step 1: A small modification is required to CarController.cs. veloKmh must be made public:

Step 1: A small modifation is required to CarController.cs. veloKmh must be made public:

```
float veloKmh; ➔ public float veloKmh;
```

Step 2: Import the FFB controller package into a UnityCar project.

Step3: The FFB Controller package includes a file UnityCar.zip. Unzip the contents of this into your project. These can be placed in any folder except a folder in or under Standard Assets.

Step 4: Copy FFWheelInput.dll from FFB Folder to root folder of project (i.e. parent folder of Assets)

Step 5: Add WheelMenu.cs to Camera Main.

Step 6: Add FFB.cs to vehicle

Step 7: On vehicle enable forcefeedback on carDynamics script

When you now drive the vehicle you should have force feedback playing through your game controller.

INSTRUCTIONS FOR ADDING DIRECT INPUT

Step 1: If you wish to use direct input: To use direct input and map keys using the FFB menu (displayed when you press "M") You can either:

a. Add option to carDynamics.cs to select direct input controller, DinputCarController. I have not supplied code to do this.
b. Or, an easier options is in carDynamics.cs, just replace references to AxisCarController class with DinputCarController.
    i. This occurs in 4 places. *Note, this is every occurace of "AxisCarController" with a capital "A". "axisCarController" with a lower case "a" is the variable name, this does not need to be changed unless you want to for aesthetic reasons. See Below for details.*
    ii. You must then add DinputCarController to the vehicle and remove AxisCarController from the vehicle.

Step 2: Once you have completed these steps your input will bypass untiy's input system and be retrieved directly from the game controller using dinput. You can use the WheelMenu GUI menu activated by pressing "M" to map your controllers input.

Further details on 1.b.i above:

Replacing references to AxisCarController with DinputCarController.

In Class Declarations:

```
AxisCarController axisCarController;
```

➔

```
DinputCarController axisCarController;
```

In Awake method:

```
axisCarController = GetComponent <AxisCarController>();
```

➔

```
axisCarController = GetComponent <DinputCarController>();
```

In SetController, 2 places:

```
axisCarController = GetComponent <AxisCarController>(); // to make this function works with SaveSetup()
```

➔

```
axisCarController = GetComponent <DinputCarController>(); // to make this function works with SaveSetup()
```

and

```
axisCarController=transform.gameObject.AddComponent<AxisCarController>();
```

➔

```
axisCarController=transform.gameObject.AddComponent<DinputCarController>();
```