# Remote Control Car Tutorial

* [1 - Background](#StepsforKartingTutorial-1-Background)
  + [Definition:](#StepsforKartingTutorial-Definition:)
* [2- Importing and Setting up the Project](#StepsforKartingTutorial-2-ImportingandS)
* [3 - Set up the Base Scene](#StepsforKartingTutorial-3-SetuptheBaseS)
* [4 - Create a Input Action Map](#StepsforKartingTutorial-4-CreateaInputA)
* [5 - Create Input Controllers](#StepsforKartingTutorial-5-CreateInputCo)
* [6 - Creating a Player Controller](#StepsforKartingTutorial-6-CreatingaPlay)
* [7 - Creating the Player Movement](#StepsforKartingTutorial-7-CreatingthePl)
* [8 - Creating Tire Marks](#StepsforKartingTutorial-8-CreatingTireM)

## ****1 - Background****

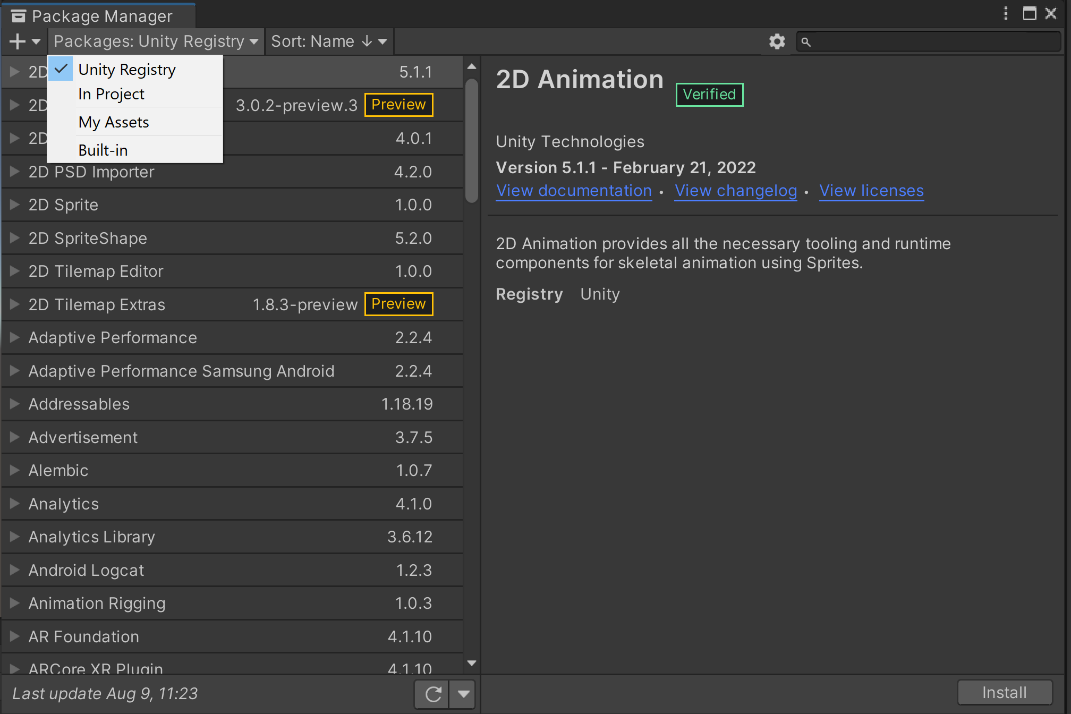
This Remote Control Car Tutorial was conducted during the U A Little Rock’s Emerging Analytics Center (EAC) visit to

### Definition:

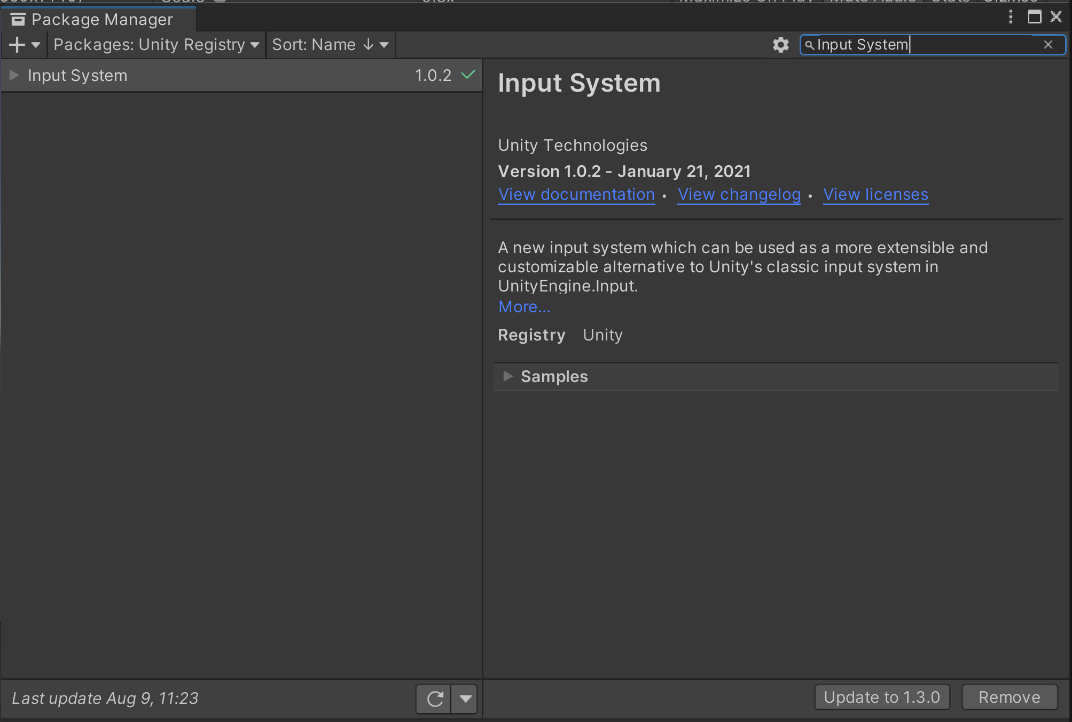
1. Vector:
   1. A property that has both a magnitude & a direction
2. Drag:
   1. A force that tends to slow the movement of an object through a liquid or gas.
3. Magnitude:
   1. The maximum extent of size and direction of an object
4. Normalized:
   1. Makes the vector have a magnitude of 1.

## ****2- Importing and Setting up the Project****

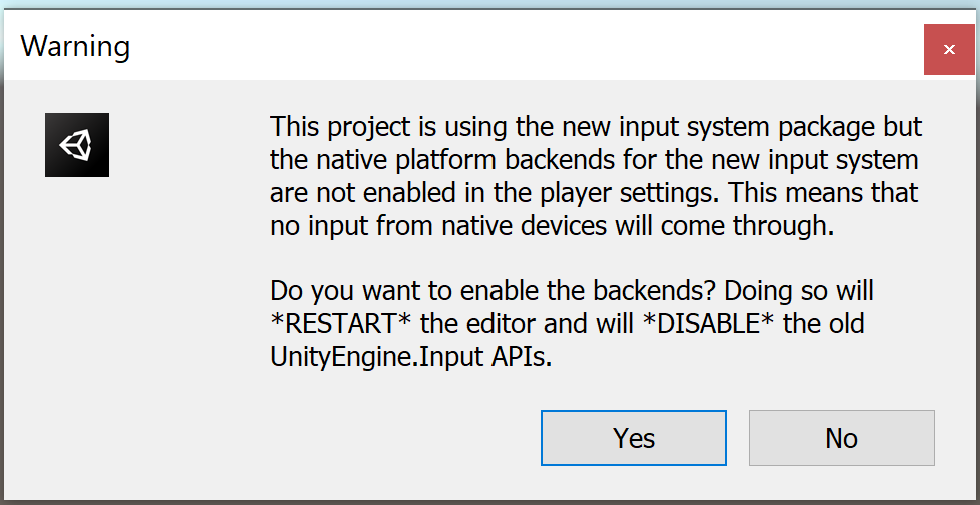
1. Open a new project
2. Modify the Project Settings:
3. Make modifications within the Package Manager:
   1. Change the Package from “In Project” to “Unity Registry”



* 1. Install the Input Systems
     1. Search for Input System and install



* + 1. When a warning dialog box pops up, select yes.



## ****3 - Set up the Base Scene****

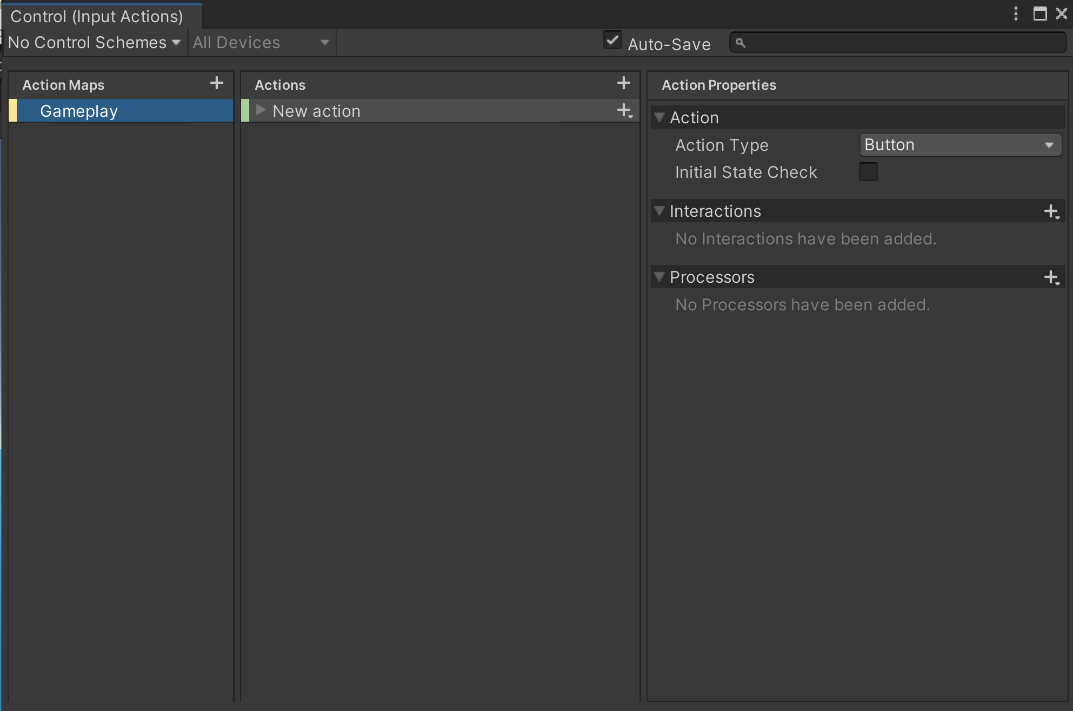
As of January 2023, all of the following Assets can be found free on the Unity Asset Store.

1. Import the assets into the project:
   1. ARCADE: Free Racing Car
   2. HQ Racing Car Model No.1203
   3. Fantasy Skybox FREE
   4. Modular Lowpoly Track Roads FREE
2. Save the scene as “Remote Control Car Game”
3. Add the following prefabs to your scene:
   1. Plane
   2. Cube

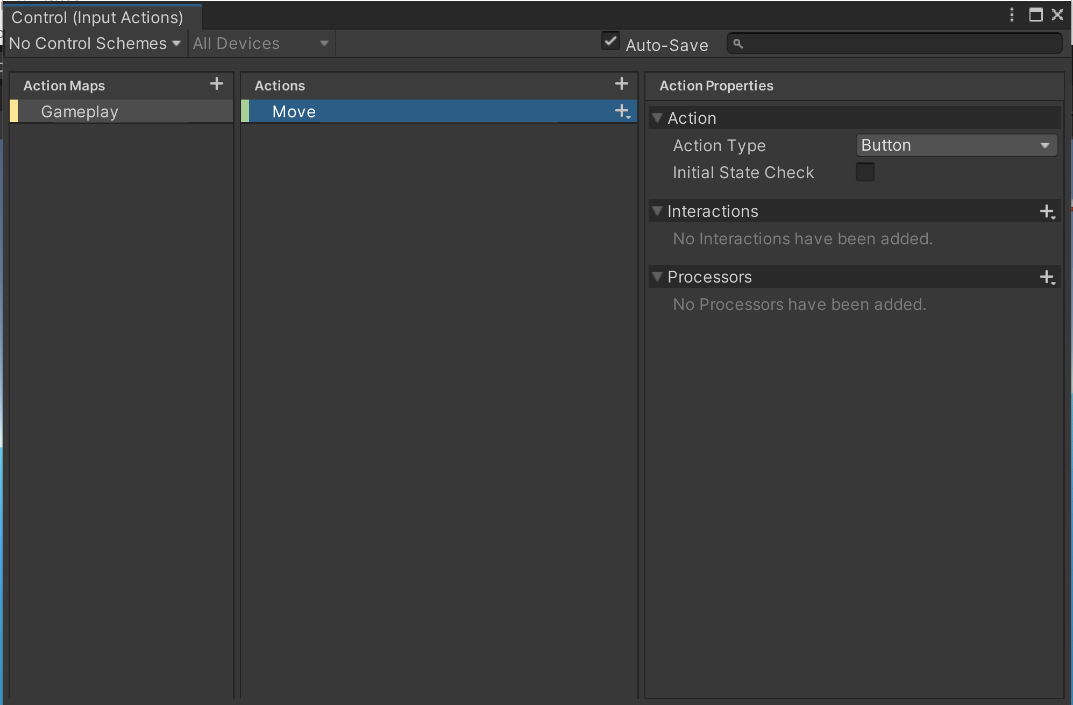
## ****4 - Create a Input Action Map****

Reasoning: The Input Action Map is used to connect the actions that are done on a input device to generate information that is used to create an action on screen. (i.e. Depending on the amount of time the up arrow on a keyboard is pressed, the distance that a character is going to move increases.)

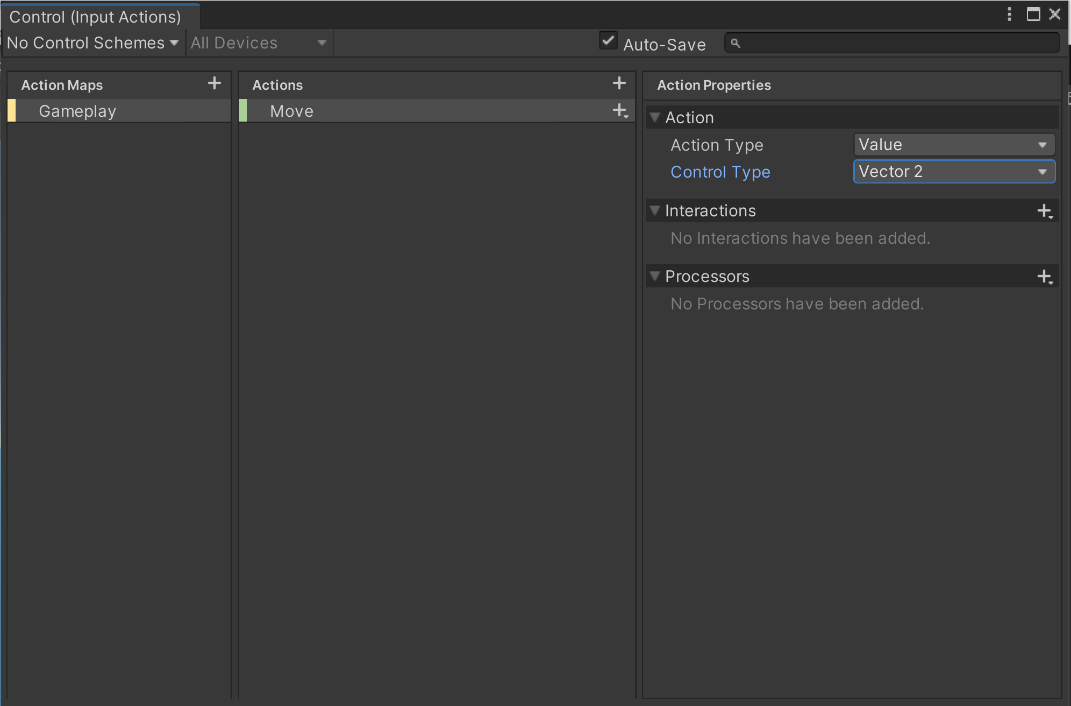
1. Create an Input Action in the asset panel and label it Controls
2. Open the Control Input Action.
3. Check Auto-Save
4. Add a Action Map and label it Gameplay\*



1. Add Action, binding, and properties
   1. Change the “New Action” to “Move”
      1. delete the default binding

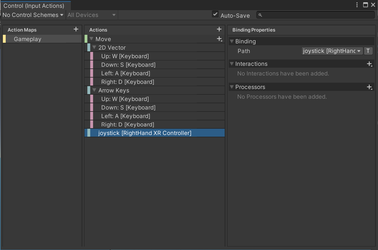


* 1. Change the Move Action properties:
     1. Change the action type to Value
     2. Change the Control Type to Vector 2

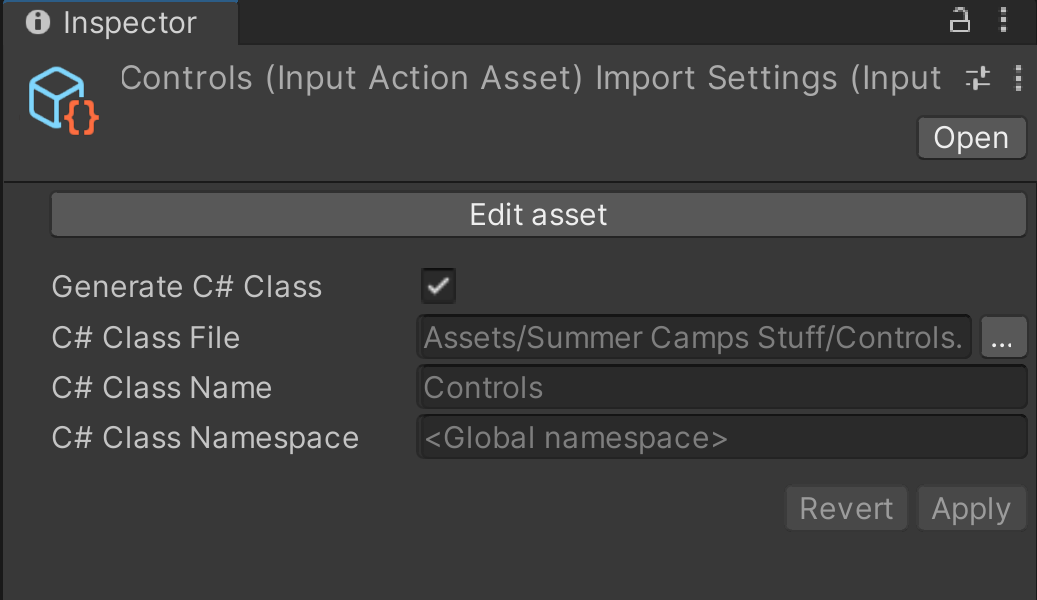


* 1. Select the plus sign to the right of the Move Action and add a (Up/Down/Left/Right) Composite
  2. Rename the composite “WASD”
  3. Change the path for all four directions:
     1. Change the path for Up to W[Keyboard]
     2. Change the path for Down to S[Keyboard]
     3. Change the path for Left to A[Keyboard]
     4. Change the path for Right to D[Keyboard]
  4. Add another (Up/Down/Left/Right) Composite to the Move
  5. Rename the composite “Arrow Keys”
  6. Change the path for all four directions:
     1. Change the path for Up to W[Keyboard]
     2. Change the path for Down to S[Keyboard]
     3. Change the path for Left to A[Keyboard]
     4. Change the path for Right to D[Keyboard]
  7. Add a binding to the Move
  8. Change the path to joystick [RightHand XR Controller]

1. Review and close the Controls Input Action Map



1. Check Generate C# Class and press Apply\*\*



\*If creating a complete game with a UI interface, you would also create two action maps. One for gameplay and another for Game Interface.

\*\*Anytime you make changes to any of the Input Action Maps in the Controls Input Actions, the C# Class is automatically going to regenerate.

## ****5 - Create Input Controllers****

Reasoning: Input Controller is what is used to delineate which Input Map is being enabled or disabled. Also, for the actions that were defined within our Input Maps, recognize when a action is started, performed, and cancelled. Also, read the direction information that was received for the input.

1. Create C# Script and label it “InputController”
2. Add the following script:

1. using System;

2. using System.Collections;

3. using System.Collections.Generic;

4. using UnityEngine;

5. using UnityEngine.Events;

6. using UnityEngine.InputSystem;

7.

8. [Serializable]

9.

10. public class MoveInputEvent : UnityEvent<float,float> { }

11. public class InputController : MonoBehaviour

12. {

13. Controls controls;

14. public MoveInputEvent moveInputEvent;

15.

16. //set up anything specific to the Input Controller class

17. private void Awake()

18. {

19. controls = new Controls();

20. }

21.

22. private void OnEnable()

23. {

24. controls.Gameplay.Enable();

25. controls.Gameplay.Move.performed += OnMovePerformed;

26. controls.Gameplay.Move.canceled += OnMovePerformed;

27. }

28.

29. private void OnMovePerformed(InputAction.CallbackContext context)

30. {

31. //we know we are getting a Vector 2 and it's getting read from the context.

32. Vector2 moveInput = context.ReadValue<Vector2>();

33.

34. moveInputEvent.Invoke(moveInput.x, moveInput.y);

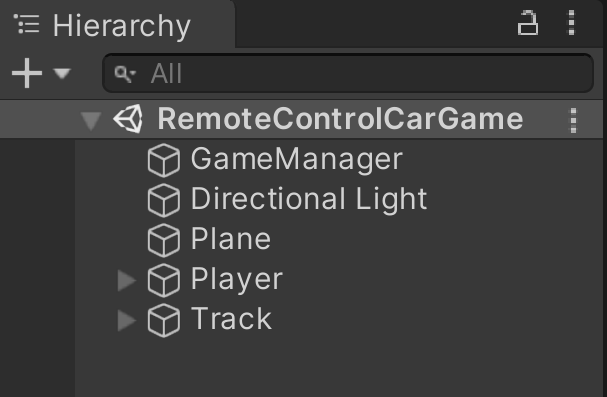
35. //Debug.Log($"Move Input:{moveInput}");

36. }

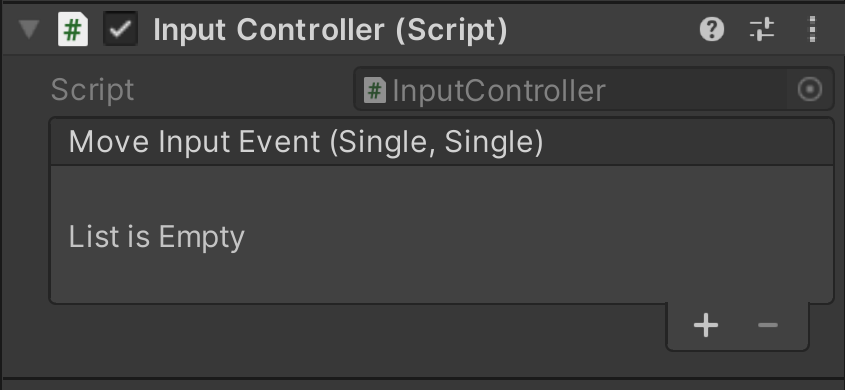
37. }

38.

1. Add an empty gameObject and rename it GameManager



1. Drag and drop InputController script onto the GameManager



## ****6 - Creating a Player Controller****

Reasoning: The player controller specifies what you actually want the car to do in your game.

1. Create a C# code and label it “CarController”
2. Add the following code:

1. using System.Collections;

2. using System.Collections.Generic;

3. using UnityEngine;

4.

5. [RequireComponent(typeof(CharacterController))]

6. public class CarController : MonoBehaviour

7. {

8. CharacterController character;

9.

10. public float MoveSpeed = 50;

11. public float MaxSpeed = 15;

12. public float Drag = 0.98f;

13. public float SteerAngle = 20;

14.

15. float horizontal;

16. float vertical;

17.

18.

19. private Vector3 MoveForce;

20.

21. private void Start()

22. {

23. character = GetComponent<CharacterController>();

24. }

25. private void Update()

26. {

27.

28. }

29. public void onMoveInput(float horizontal, float vertical)

30. {

31. this.horizontal = horizontal;

32. this.vertical = vertical;

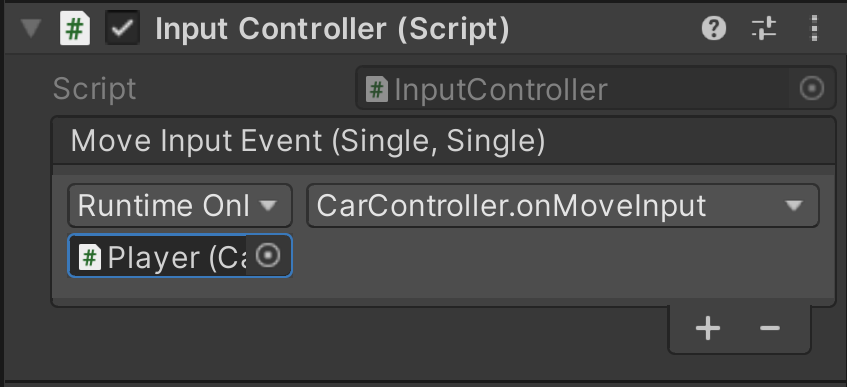
33. Debug.Log($"Player controller:New Input: {horizontal},{vertical}");

34. }

35. }

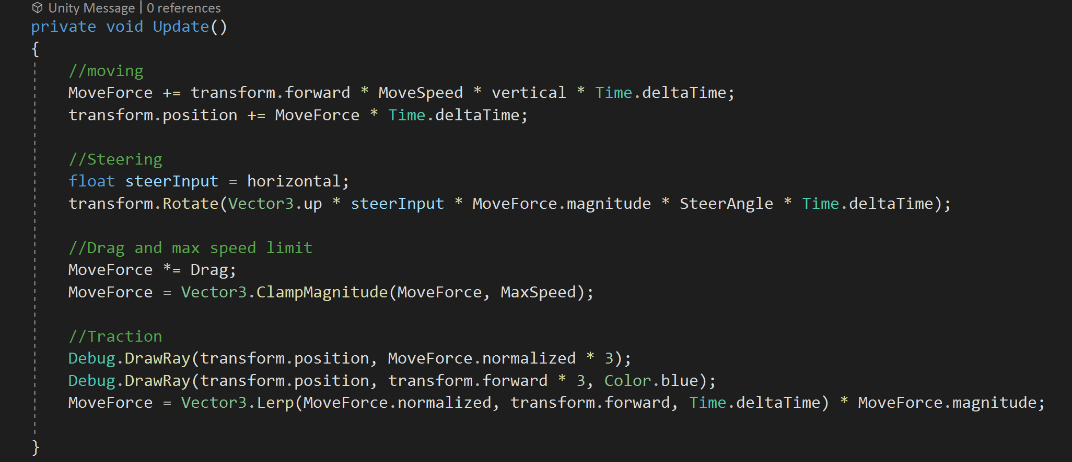
36.

1. Save the code and close the code editor to open the Unity interface
2. Drag and drop the CarController script on the Player in the Hierarchy panel.
3. Select the GameManager in the Hierarchy panel
   1. The properties for the GameManager should show up in the Inspector panel.
4. Press the plus sign at the bottom on the Move Input Event (Single, Single)
5. Add the player and PlayerController.onMoveInput method to the Move Input Event



## ****7 - Creating the Player Movement****

Reasoning: This is where we defined the type of movement that we would like the car to do.



1. Add the following code segments to the Update function in the CarController script
   1. Moving

1. MoveForce += transform.forward \* MoveSpeed \* vertical \* Time.deltaTime;

2. transform.position += MoveForce \* Time.deltaTime;

3.

* 1. Steering

1. float steerInput = horizontal;

2. transform.Rotate(Vector3.up \* steerInput \* MoveForce.magnitude \* SteerAngle \* Time.deltaTime);

3.

* 1. Drag and max speed limit

1. MoveForce \*= Drag;

2. MoveForce = Vector3.ClampMagnitude(MoveForce, MaxSpeed);

3.

* 1. Traction

1. Debug.DrawRay(transform.position, MoveForce.normalized \* 3);

2. Debug.DrawRay(transform.position, transform.forward \* 3, Color.blue);

3. MoveForce = Vector3.Lerp(MoveForce.normalized, transform.forward, Time.deltaTime) \* MoveForce.magnitude;

4.

1. Save the code