Tiny Car Controller

User manual v1.5.0

Table of Contents

Description

This controller allows you to create and control a basic vehicle with fully configurable, arcade-like physics.

Instead of having a body with four separate wheels, which is often needlessly complicated and prone to glitches, this controller is composed of a single rigidbody and sphere collider, similar to a character controller.

Features

- Easy one step setup
- Control acceleration, speed, drifting, slope limit, collisions, and more
- Change surfaces parameters to create for instance ice, mud, etc.
- Lightweight, perfect for mobile games
- Includes example scripts to take care of input, visuals, and camera
- No risk of flipping over
- Can be used for any type of vehicle

Requirements

Unity version 2018.4 or later is recommended, but it should also work with any version of the 2018.x family.

Intermediate C# programming knowledge is strongly advised.

Contact

David Jalbert (programmer)
Email: jalbert.d@hotmail.com

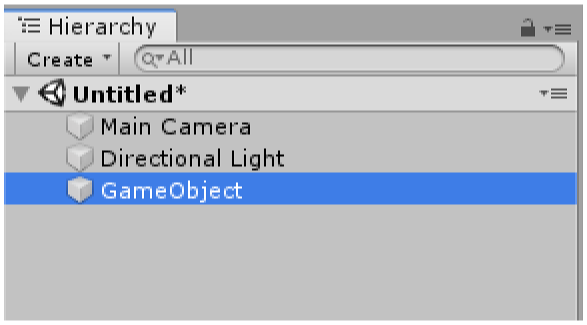
Unity asset store package
https://assetstore.unity.com/packages/tools/physics/tiny-car-controller-151827

Unity forums official thread
https://forum.unity.com/threads/tiny-car-controller-official-thread.852850/

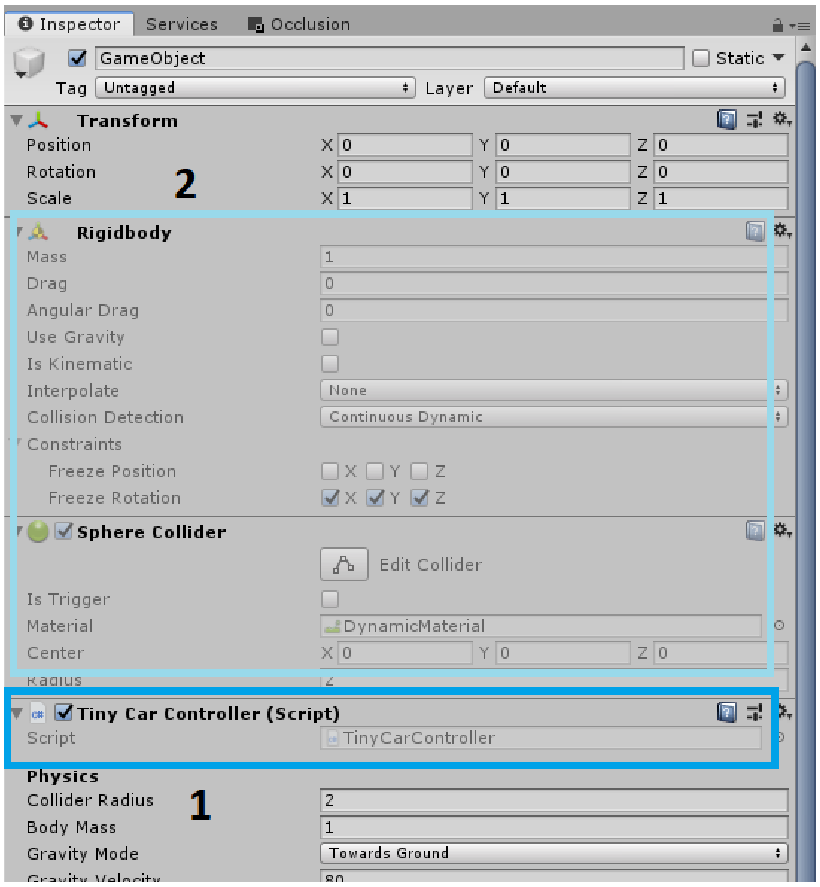
Getting started

Minimal setup

1) Create an empty GameObject in your scene.



2) Add the script at 'Assets/DavidJalbert/TinyCarController/Components/TinyCarController.cs' to the GameObject.



GameObject ✓ Static
Tag Untagged Layer Default

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

Rigidbody
Mass 1
Drag 0
Angular Drag 0
Use Gravity ☐
Is Kinematic ☐
Interpolate None
Collision Detection Continuous Dynamic
Constraints
Freeze Position ☐ X ☐ Y ☐ Z
Freeze Rotation ✓ X ✓ Y ✓ Z

✓ Sphere Collider
Edit Collider
Is Trigger ☐
Material DynamicMaterial
Center X 0 Y 0 Z 0
Radius 2

✓ Tiny Car Controller (Script)
Script TinyCarController

Physics
Collider Radius 2
Body Mass 1
Gravity Mode Towards Ground
Gravity Velocity 80

When the component is added to an empty GameObject (1), a read-only Rigidbody and Sphere Collider will be created (2). You don’t need to modify their values directly, they’re only used by the controller.

3) Adjust the parameters in the 'Tiny Car Controller' component to fit the desired size, mass, and behavior.

At this point, your controller is ready to use, but it will not have any graphics or input. To do that, you need additional scripts, some of which are used in the example scene.

Parameters

Physics

Collider Radius

The radius of the controller’s sphere collider. It will be automatically applied once it’s changed.

Body Mass

The mass of the controller’s rigidbody. It will be automatically applied once it’s changed.

Gravity Mode

Changes the way the gravity works. “Always down” will make the gravity always point down even on slopes; “Towards ground” will make the gravity point to the surface of a stable slope when grounded.

Gravity Velocity

The speed at which the controller will accelerate towards the direction of the gravity.

Max Gravity

The maximum speed at which the controller will move towards the direction of the gravity.

Max Slope Angle

The maximum angle at which the controller can climb a slope. If the angle of the slope is higher than this number, the controller will slide down.

Side Friction

The amount of friction to add to the controller when driving along a wall.

Collidable Layers

The layers that will be used to detect ground. This is only used for ground detection; you need to also set the gameobject’s layer accordingly if you need it to ignore all collisions with specific objects.

Engine

Acceleration Curve

The multiplier to apply to the controller’s acceleration depending on its speed. Values on both axes must be between 0 and 1 to work properly.

Max Acceleration Forward

The maximum acceleration at which the controller can go going forward.

Max Speed Forward

The maximum speed at which the controller can go going forward.

Max Acceleration Reverse

The maximum acceleration at which the controller can go going backwards.

Max Speed Reverse

The maximum speed at which the controller can go going backwards.

Brake Strength

The force to apply to the controller when accelerating in the opposite direction that it’s going.

Slope Friction

The amount of friction to apply when going up a slope. The actual amount of friction will depend on the steepness of the slope. This can be used to simulate torque.

Steering

Max Steering

The speed at which the controller will turn when applying steering.

Steering Multiplier In Air

The multiplier to apply to the steering when the controller is ungrounded.

Steering By Speed

The multiplier to apply to the steering depending on the speed of the controller. Values on both axes must be between 0 and 1 to work properly.

Forward Friction

The amount of friction to add to the controller’s forward velocity when grounded.

Lateral Friction

The amount of friction to add to the controller’s side velocity when grounded. The lower the friction, the more the controller will drift.

Helper Scripts

Standard Input

The script “TinyCarStandardInput” takes care of keyboard and gamepad controls.

Add the script to any GameObject and change the parameters to your needs.

The keys, buttons, and axes that will be used to control the car are defined in the “Input” section. The values used are the same as in Unity’s Input class. If you select the type “Key”, the name of the key should be a numerical value corresponding to Unity’s KeyCode enum.

Mobile Input

The script “TinyCarMobileInput” is used to control the vehicle through the touch screen.

Add the script to any GameObject and change the parameters to your needs. You also need an EventSystem component somewhere on your scene for it to work.

The “Steering Wheel”, unlike the other UI elements, has a separate “Touch Area” and “Image” object. The “Touch Area” is the UI element that will be used to register touch events for the steering wheel. Ideally this should be an Image component with “Raycast Target” checked and its color alpha set to zero. The “Image” object is the graphic of the steering wheel itself. It will be rotated by 90 degrees to the left and right depending on where the Touch Area is touched.

The “Steering Wheel Multiplier” is by how much to multiply the steering value relative to the touch area. This can be useful if you want to have a large touch area that clamp the steering to its max value when touching the edges.

Visuals

The script “TinyCarVisuals” synchronizes the controller with the graphics of the vehicle.

Ideally you want this component in its own GameObject, so that if you disable either the controller or visuals, the script still keeps running. In the Example scene, this component is in the “Car Container” object.

Camera

The script “TinyCarCamera” synchronizes the controller with a single camera object.

This component should be added to a GameObject containing a Camera component. It will follow the target while it is enabled.

Explosive Body

The script “TinyCarExplosiveBody” is used to make the vehicle blow up on command.

Ideally you want this component in its own GameObject, so that if you disable the controller, the script still keeps running. In the Example scene, this component is in the “Car Container” object.

Essentially, when you call the “explode()” function, this script turns off the controller and visuals, turns on the “Parts Container” object, and adds force and torque to all its children that contain a Rigidbody component. When calling the “restore()” function, it turns on the controller and visuals, turns off the “Parts Container” object, and sets the controller’s position and rotation to that of the “Spawn Point” transform.

Surface

The script “TinyCarSurface” changes the friction and speed values of the vehicle when they come in contact with each other.

You can either add this script to a static collider, or to a trigger collider. When coming in contact with an object containing the script, the controller will change its steering, speed, and acceleration according to the surface’s parameters.

Functions Reference

Below are all the public functions you can use in the TinyCarController script with a short description for each.

void clearVelocity()

Resets the controller's velocity to zero.

Rigidbody getBody()

Returns the controller’s Rigidbody component.

Vector3 getBodyPosition()

Returns the position of the controller’s Rigidbody.

Quaternion getBodyRotation()

Returns the rotation of the controller’s Rigidbody.

float getBoostMultiplier()

Returns the current value of the boost multiplier.

float getForwardVelocity()

The velocity of the controller going forward in units per second.

float getForwardVelocityDelta()

The velocity of the controller going forward relative to its maximum speed. Returns a value between -1 and 1.

Vector3 getGravityDirection()

Returns a normalized Vector3 representing the direction of the controller’s gravity.

float getGroundHitForce()

Returns the relative velocity of the collision with the ground during the current frame.

Vector3 getGroundPosition()

Returns the position of the controller at ground level.

Quaternion getGroundRotation()

Returns the angle of the ground represented as a Quaternion.

float getGroundVelocity()

Returns the velocity of the controller on the XZ plane represented in units per second.

float getGroundVelocityDelta()

The velocity of the controller on the XZ plane relative to its maximum speed. Returns a value between -1 and 1.

float getLateralVelocity()

Returns the velocity of the controller relative to its X axis, represented in units per second.

float getMaxAcceleration()

Returns the controller’s maximum acceleration value.

float getMaxSpeed()

Returns the controller’s maximum speed value.

float getMotor()

The current input to the controller’s motor. Returns a value between -1 and 1.

int getMotorDirection()

The direction of the controller’s motor. Returns an integer value between -1 and 1.

float getSideHitForce()

Returns the current frame’s collision force value on the XZ plane.

Vector3 getSideHitPosition()

Returns the current frame’s collision position on the XZ plane.

float getSlopeDelta()

How much the angle of the ground affects the velocity of the controller. Returns a value between -1 and 1, where -1 is going down at 90 degrees, and 1 is going up at 90 degrees.

float getSteering()

The current input to the controller’s steering. Returns a value between -1 and 1.

TinyCarSurfaceParameters getSurfaceParameters()

Returns the current ground surface parameters as a TinyCarSurfaceParameters object.

int getVelocityDirection()

The direction of the controller’s forward velocity. Returns -1 if going backward, 0 if stationary, and 1 if going forward.

bool hasHitGround(float minDownwardVelocity = 0)

Returns whether the controller has hit the ground this frame. Can take the minimum velocity to consider as a parameter.

bool hasHitSide(float minForce = 0)

Returns whether the controller has hit a wall this frame. Can take the minimum velocity to consider as a parameter. Only returns true if the controller was not hitting a wall during the previous frame.

bool isBraking()

Returns whether the controller is braking.

bool isGrounded()

Returns whether the controller is on stable ground.

bool isHittingSide()

Returns whether the controller is colliding with a wall.

void setBoostMultiplier(float m)

Sets the controller’s boost multiplier.

void setMotor(float value)

Sets the controller’s motor input. Must be a value between -1 and 1.

void setSteering(float value)

Sets the controller’s steering input. Must be a value between -1 and 1.

Troubleshooting

Can I use different colliders than the default sphere?

You can technically add any number of secondary colliders to the controller and they should work properly. Keep in mind that only the main sphere collider is used to detect ground.

How can I detect when the car collides with a custom object?

You can add secondary scripts to the controller and use functions like “OnTriggerEnter” and “OnCollisionEnter” to detect collisions.

The car jumps a bit when running over seams in the road. How can I fix that?

This is unfortunately a downside of the method used to program this controller. You can try changing the values of the “Default Contact Offset” parameter in the Physics tab of the Project Settings to minimize the effects of “ghost collisions”. Ideally, this controller works best on connected meshes.