Tiny Car Controller

User manual v2.0

Table of Contents

Description

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

Instead of using Unity's default wheel collider, which is often needlessly complicated and prone to glitches, the wheels on this controller are made of sphere colliders with several specifically configured joints to mimic a vehicle's motor, steering, and suspension.

Features

- Easy hassle-free setup
- Control acceleration, speed, friction, suspension, collisions, and more
- Uses Unity's physics engine for perfect compatibility with other assets
- Lightweight, perfect for mobile games
- Includes example scripts to take care of input and camera
- Compatible with any OS, render pipeline, or Unity version

Requirements

Unity version 6.0 or later is recommended, but it should also work with older versions.

Intermediate C# programming knowledge is strongly advised.

Contact

David Jalbert (programmer)

Email: davidjalbertgames@outlook.com

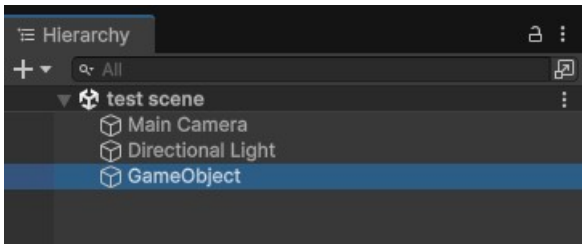
Bsky: https://bsky.app/profile/davidjalbertgames.ca

Unity asset store package: https://u3d.as/1AY1

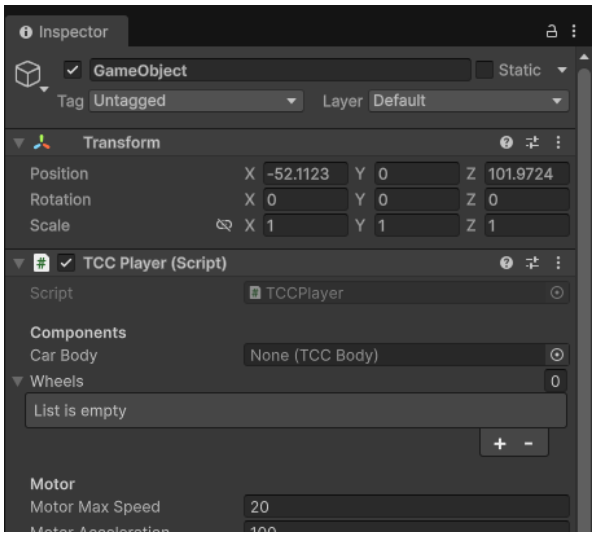
WebGL demo: https://davidjalbert.itch.io/tiny-car-controller-webgl-demo

Minimal setup

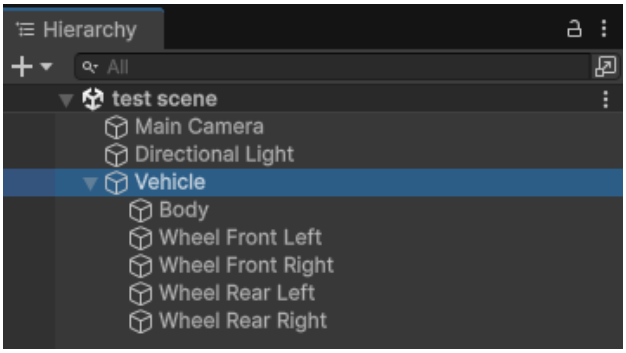
1) Create an empty GameObject in your scene.



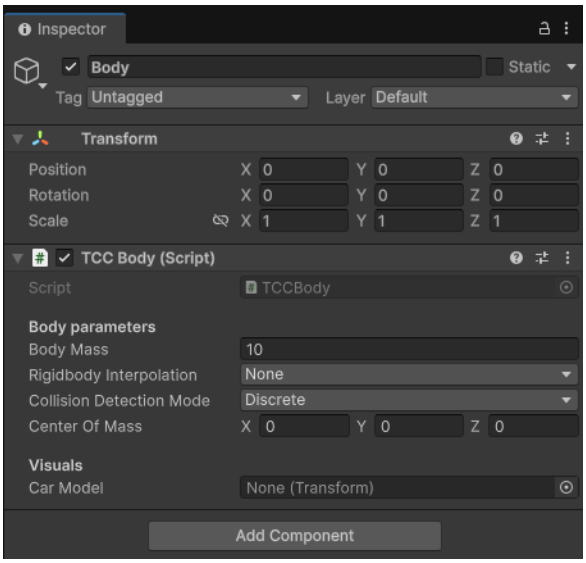
2) Add the script at “Assets/DavidJalbert/TinyCarController/Scripts/Core/TCCPlayer.cs” to the empty GameObject



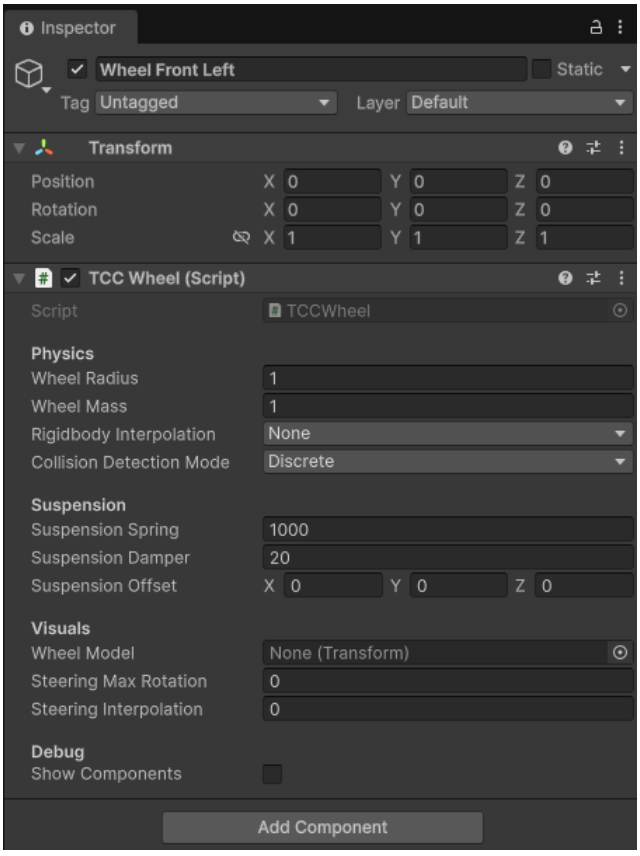
3) Create empty GameObjects as children of the root GameObject for the body of the vehicle and for each wheel.



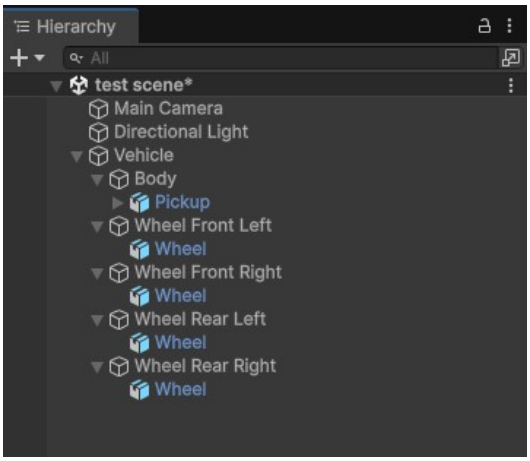
4) Add the script at 'Assets/DavidJalbert/TinyCarController/Scripts/Core/TCCBody.cs' to the object that will serve as the vehicle body.



5) Add the script at 'Assets/DavidJalbert/TinyCarController/Scripts/Core/TCCWheel.cs' to each of the objects you created to serve as the vehicle wheels.



6) Add your models as children of the corresponding GameObjects. Make sure you have one object per wheel and one for the body, and that they are separated.



7) The pivot of your body and wheel objects will determine how the spring joints will be configured, so make sure to position them accordingly. The models should be centered as much as possible with the pivot of their parent object.

8) Set the parameters of the body and wheel objects to your liking. The default parameters are generally good enough.

9) Set the parameters for the vehicle in the TCCPlayer script that you added to the root object. This is what will control things like max speed, acceleration, tire grip, and roll and pitch countering.

At this point, your controller is ready to use, but it will not have any input or camera control. To do that, you need additional scripts, which are used in the example scene. Check out the scene at “DavidJalbert/TinyCarController/Examples/ExampleSandbox” to see how to use them.

Parameters

TCC Player

Components

Car Body

Object containing the TCCBody script. If empty, will search for a suitable child.

Wheels

Objects containing the TCCWheel scripts. If empty, will search for suitable children.

Motor

Motor Max Speed

The maximum speed at which the car should go.

Motor Acceleration

The base acceleration speed of the car.

Brake Force

How fast the car should slow down when braking.

Motor Input Smoothing

How much smoothing to apply to the motor input. Set to zero to change the motor input instantly.

Brake Input Smoothing

How much smoothing to apply to the brake input. Set to zero to change the brake input instantly.

Speed Over Max Damping

How fast the car should slow down when its speed is above the maximum speed.

Acceleration Over Speed

The multiplier to apply to the acceleration (Y axis) depending on the current speed of the car relative to its maximum speed (X axis).

Speed Ground Damping

How quickly the car will slow down when the motor is not accelerating.

Speed Air Damping

How quickly the car will slow down on its local Z axis when in the air.

Downhill Max Speed Multiplier

The multiplier to apply to the max speed when going downhill. Will be interpolated between 0 and 90 degrees.

Downhill Acceleration Multiplier

The multiplier to apply to the acceleration when going downhill. Will be interpolated between 0 and 90 degrees.

Uphill Max Speed Multiplier

The multiplier to apply to the max speed when going uphill. Will be interpolated between 0 and 90 degrees.

Uphill Acceleration Multiplier

The multiplier to apply to the acceleration when going uphill. Will be interpolated between 0 and 90 degrees.

Steering

Steering Input Smoothing

How much smoothing to apply to the steering input. Set to zero to change the steering input instantly.

Steering Speed

How fast the car will turn when steering.

Steering Air Speed

How fast the car will turn when steering in the air.

Steering Min Speed

The minimum speed of the car at which it can steer fully. Below this value, the car will steer progressively slower as the speed goes down.

Steering Air Damping

How fast to slow down the car rotation while in the air.

Tire Grip

How much the wheels will stick to the surface. Lower values will make the car drift more.

Tire Grip Over Speed

The multiplier to apply to the tire grip (Y axis) depending on the current speed of the car relative to its maximum speed (X axis).

Drift Min Velocity

Minimum speed at which the car will move laterally (drifting) before it's set to zero. Helps stabilize the car on slippery surfaces.

Drift Air Damping

How quickly the car will slow down on its local X axis when in the air.

Roll Countering

Roll Counter Mode

When the car should stabilize itself by countering its roll angle. Set the parameter to Never to disable this feature.

Roll Counter Speed

The speed at which the car should turn when countering its roll angle.

Roll Counter Acceleration

How fast the car will reach its maximum turn speed when countering its roll angle.

Roll Counter Angle

The angle at which the car will try to set itself when countering its roll angle.

Pitch Countering

Pitch Counter Mode

When the car should stabilize itself by countering its pitch angle. Set the parameter to Never to disable this feature.

Pitch Counter Speed

The speed at which the car should turn when countering its pitch angle.

Pitch Counter Acceleration

How fast the car will reach its maximum turn speed when countering its pitch angle.

Pitch Counter Angle

The angle at which the car will try to set itself when countering its pitch angle.

Modifiers

Max Speed Multiplier

Multiplies the max speed of the car by this value. Useful to simulate a temporary boost.

Acceleration Multiplier

Multiplies the acceleration of the car by this value.

Tire Grip Multiplier

Multiplies the tire grip of the car by this value.

TCC Body

Body Parameters

Body Mass

The mass that will be applied to the body.

Rigidbody Interpolation

Whether to apply interpolation to the body. Set this parameter to Interpolate to prevent jittering when moving.

Collision Detection Mode

Which collision detection mode to use on the body. Set this parameter to Continuous Dynamic to avoid going through other objects at high speed, or getting stuck.

Center Of Mass

The center of mass of the body in local space. This parameter changes how the rigidbody behaves when not on stable ground. Ideally this should be the center of the car at ground level.

Visuals

Car Model

The object containing the car model. Will use the first child object if this field is empty.

TCC Wheel

Physics

Wheel Radius

Radius of the wheel collider.

Wheel Mass

Mass of the wheel rigidbody. Should be lower than the mass of the car body.

Rigidbody Interpolation

Whether to use interpolation for the wheel rigidbody. Set this parameter to Interpolate to prevent jittering when moving.

Collision Detection Mode

Which collision detection mode to use for the wheel collider. Set this parameter to Continuous Dynamic to avoid going through other objects at high speed, or getting stuck.

Suspension

Suspension Spring

Force applied to the suspension. Higher values make the suspension stiffer.

Suspension Damper

Damper applied to the suspension. Higher values make the suspension settle faster.

Suspension Offset

Shifts the position of the wheel relative to its initial position. Useful to mimic hydraulics.

Visuals

Wheel Model

The object containing the wheel model. Will be locally rotated according to steering and velocity. Will use the first child object if this field is empty.

Steering Max Rotation

The maximum angle in degrees by which the wheel will turn when steering. A negative value will turn the wheel in the opposite direction.

Steering Interpolation

The speed at which the wheel will turn when steering. Set to zero to disable interpolation.

Debug

Show Components

Show or hide the generated components in the inspector when running the game.

TCC Surface

Max Speed Modifiers

By how much to multiply the car's max speed when driving over this surface.

Acceleration Modifier

By how much to multiply the car's acceleration when driving over this surface.

Tire Grip Modifier

By how much to multiply the car's tire grip when driving over this surface.

Scripts Reference

TCCPlayer

Initializes and controls the behavior of the wheels and body, and includes functions to set and return the position and rotation of the vehicle, check the grounding state, velocities, etc.

`public TCCBody getCarBody()`

Returns the TCCBody component associated with this object.

`public Rigidbody getCarRigidBody()`

Returns the Rigidbody component associated with this object.

`public TCCSurfaceModifier getSurfaceModifier()`

Returns a TCCSurfaceModifier object representing the average surface values this vehicle is currently standing on.

`public Vector3 getVelocity()`

Returns the linear velocity of this object in world space.

`public float getForwardVelocity()`

Returns the velocity of this object on its local Z axis.

`public float getRightVelocity()`

Returns the velocity of this object on its local X axis.

`public float getUpVelocity()`

Returns the velocity of this object on its local Y axis.

`public float getGroundVelocity()`

Returns the velocity of this object on the XZ plane in world space.

`public float getPitchAngle()`

Returns the angle in degrees of the vehicle on its local X axis.

`public float getRollAngle()`

Returns the angle in degrees of the vehicle on its local Z axis.

`public float getPitchVelocity()`

Returns the angular velocity of the vehicle on its local X axis.

`public float getRollVelocity()`

Returns the angular velocity of the vehicle on its local Z axis.

`public float getYawVelocity()`

Returns the angular velocity of the vehicle on its local Y axis.

`public bool isPartiallyGrounded()`

Returns whether the vehicle has at least one wheel on the ground but not all of them.

`public bool isUngrounded()`

Returns whether the vehicle's wheels are all off the ground.

`public bool isFullyGrounded()`

Returns whether the vehicle's wheels are all on the ground.

`public void setMotor(float d)`

Sets the input of the motor between -1 and 1, where 1 is going at full speed, and -1 is going fully in reverse.

`public void setSteering(float d)`

Sets the input of the steering between -1 and 1, where -1 is fully to the left, and 1 is fully to the right. Zero is centered.

`public void setBrake(float d)`

Sets the input of the brakes between 0 and 1, where 0 is not braking at all, and 1 is fully braking.

`public float getMotor()`

Gets the current input of the motor.

`public float getSteering()`

Gets the current input of the steering.

`public float getBrake()`

Gets the current input of the brakes.

`public float getSmoothMotor()`

Gets the current input of the motor with smoothing applied.

`public float getSmoothSteering()`

Gets the current input of the steering with smoothing applied.

`public float getSmoothBrake()`

Gets the current input of the brakes with smoothing applied.

`public void immobilize()`

Sets the linear and angular velocities of the vehicle to zero.

`public Vector3 getPosition()`

Returns the position of the vehicle in world space.

`public void setPosition(Vector3 v)`

Sets the position of the vehicle in world space.

`public Quaternion getRotation()`

Returns the rotation of the vehicle in world space.

`public void setRotation(Quaternion q)`

Sets the rotation of the vehicle in world space.

`public void translate(Vector3 v)`

Moves the vehicle by the value provided.

`public void rotate(Quaternion q)`

Rotates the vehicle by the value provided.

TCCBody

Manages the vehicle's body, and includes functions to detect collisions.

Delegates

The following delegates are called during their respective MonoBehaviour functions, and can be used by assigning your own function to them with the "+=" operator. For instance, you could create a function called "callThisOnCollision(Collision c)", which you would assign to the delegate with "vehicleBody.onCollisionEnterActions += callThisOnCollision". You can also remove an assigned function with the "-=" operator.

`public Action<Collision> onCollisionStayActions`

Actions that will be called when the body is currently in a collision with another collider.

`public Action<Collision> onCollisionEnterActions`

Actions that will be called when the body enters a collision with another collider.

`public Action<Collision> onCollisionExitActions`

Actions that will be called when the body exits a collision with another collider.

`public Action<Collider> onTriggerStayActions`

Actions that will be called when the body is currently in a collision with a trigger collider.

`public Action<Collider> onTriggerEnterActions`

Actions that will be called when the body enters a collision with a trigger collider.

`public Action<Collider> onTriggerExitActions`

Actions that will be called when the body exits a collision with a trigger collider.

Functions

`public Rigidbody getRigidbody()`

Returns the Rigidbody component associated with the object.

`public void setConstraints(RigidbodyConstraints c)`

Sets the RigidbodyConstraints parameter for the object's Rigidbody. By default, this parameter is null.

`public float getPitchAngle()`

Returns the angle in degrees of the vehicle on its local X axis.

`public float getRollAngle()`

Returns the angle in degrees of the vehicle on its local Z axis.

`public TCCPlayer getParentPlayer()`

Returns the TCCPlayer object associated with this object.

`public void immobilize()`

Sets the linear and angular velocities of this object to zero.

`public Vector3 getPosition()`

Returns the position of this object in world space.

`public Quaternion getRotation()`

Returns the rotation of this object in world space.

TCCWheel

Manages the vehicle’s wheels, and includes functions to detect collisions, and whether they are on stable ground.

Delegates

This component also has delegates to detect collision, just like the TCCBody component. Please refer to the TCCBody component for more information.

Functions

`public Rigidbody getRigidbody()`

Returns the Rigidbody component associated with the object.

`public void setConstraints(RigidbodyConstraints c)`

Sets the RigidbodyConstraints parameter for the object's Rigidbody. By default, this parameter is null.

`public TCCPlayer getParentPlayer()`

Returns the TCCPlayer object associated with this object.

`public SphereCollider getCollider()`

Returns the SphereCollider associated with this object.

`public float getColliderRadius()`

Returns the radius of the SphereCollider associated with this object.

`public bool isTouchingGround()`

Returns whether the wheel is touching a surface.

`public bool isStableOnGround()`

Returns whether the wheel is touching a surface at a maximum lateral angle of 45 degrees. Will return false if the wheel is touching the side of a curb, for instance.

`public Vector3 getPosition()`

Returns the position of this object in world space.

`public Quaternion getRotation()`

Returns the rotation of this object in world space.

`public void immobilize()`

Sets the linear and angular velocities of this object to zero.

`public TCCSurfaceModifier getSurfaceModifier()`

Returns a TCCSurfaceModifier object representing the average surface values this wheel is currently standing on.

Troubleshooting

Can I use different colliders for the wheels than the default sphere?

Unfortunately, this controller has been designed to use a specific combination of rigidbodies, colliders, and joints for the wheels, so you can't use custom colliders. However, you can use any type of collider for the car body, and the wheels won't collide with it, so if you wanted for instance to have the wheels only touch the ground and not the sides of another object, you could create a collider on the car body that intersect the wheels.

How can I detect when a wheel/body collides with a custom object?

You can assign functions to the following delegates in the TCCBody and TCCWheel components, which are called at the same time as their Rigidbody counterparts;

- **`public Action<Collision> onCollisionStayActions`**
- **`public Action<Collision> onCollisionEnterActions`**
- **`public Action<Collision> onCollisionExitActions`**
- **`public Action<Collider> onTriggerStayActions`**
- **`public Action<Collider> onTriggerEnterActions`**
- **`public Action<Collider> onTriggerExitActions`**

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

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.

How do I change the position and rotation of the vehicle?

You can set the position and rotation of the vehicle with the functions "TCCPlayer.setPosition" and "TCCPlayer.setRotation". Manually setting the position and rotation of the individual components might cause some unintended glitches.

If you need to reset the vehicle completely, you might want to also call the function “TCCPlayer.immobilize()”, which will set the vehicle’s velocities to zero.

You can also use the functions “TCCPlayer.translate(Vector3)” and “TCCPlayer.rotate(Quaternion)” if you just want to add values to the current position and rotation.