Bikes and Bicycles | Unity

As the game technically allows the player to get in any vehicle I've created a bicycle system that can be reused as a bike system, but for now, we've tried to focus a little more on the bicycle side.

A model of a bicycle made with too many polygons but useful enough to be able to create the system itself.

In order to get a bike running, I needed to research to create the system, searched for free assets in the asset store to learn from, and found nothing, and after digging a little bit I found a dead end where everyone that has created a bicycle script is only allowing you to buy it. Probably because they've spent a little time creating it, and is understandable the need for funding for the time invested in some complex systems with physical behaviors. related to bikes Ik handles for ragdoll controllers etc.
[RELEASED] Simple Bicycle Physics - Unity Forum

bicycle physics - Asset Store (unity.com)

As I said anything that had a Bike on its name, had to be paid, and with no will to exchange pocket for knowledge, I decided to go in the hard way. and try it myself.

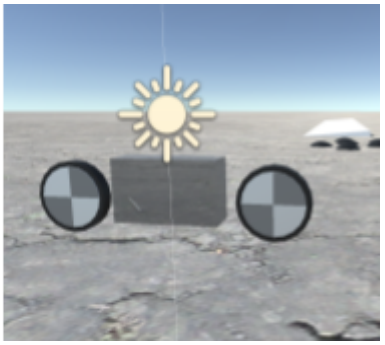
For our system, we simply needed the bikes as an extra vehicle option for the player to travel, as realistic and coherent as it can be, but as gamey as possible to make it fit in the game feel, As I already had a simple vehicle script working with the Wheelcolliders of unity I cut that in half and we had a simple system with the bike moving and braking correctly.

I realized I had to freeze rotation on the Z axis of the RB, as It does something weird if it's active. After tweaking everything the bike was running well straight up, but by no matter, we could have a bike that doesn't lean, Leaning is biking!

Bike asset Made by Myself
Rayzn - Sketchfab

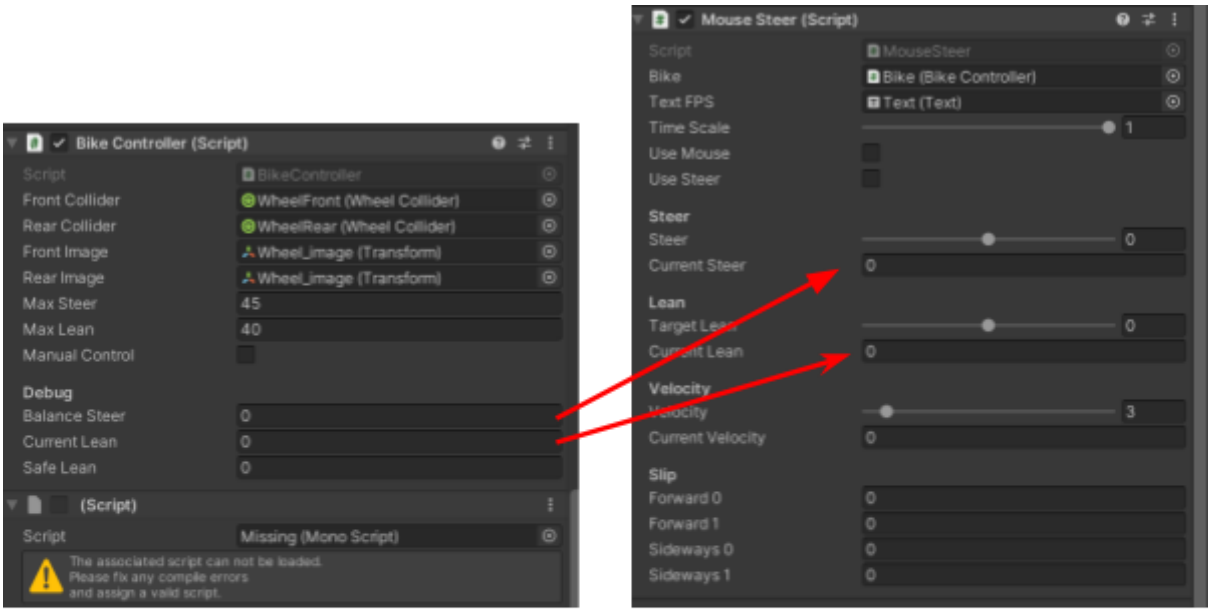
Bike Balance/Steering

Digging and researching if I could find an example asset or script that could give me a little hint going in the right direction/ Thankfully I stumbled upon a simple project exactly doing what I wanted to do but using physics



GitHub - V-Kudryashov/BikeBalance: Bike balance by steering in Unity

The way this works is a little too complicated to get around as it was physically based and accurate. But that was enough to understand the mechanic and procedure, He was using WheelColiders as well so I already had a headstart. To be clear I didn't replicate this script with such an amazing level of detail, I tell you what I got out of it.

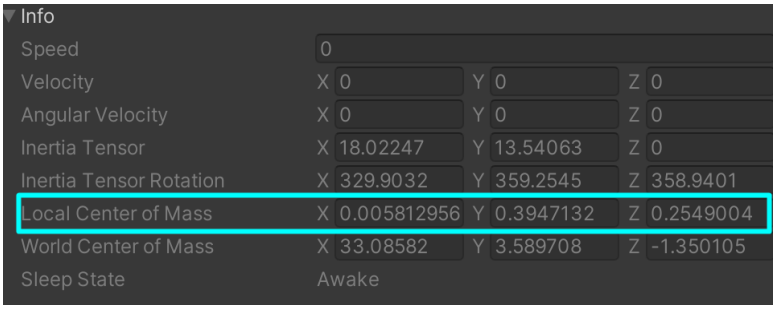


To be short in terms he basically calculated the amount of turning required to compensate for the amount of leaning that we imputed, if leaned correctly, that's calculated in relation to the velocity allowing for turning to occur smoothly and not overturn at high velocities. , but overlean and you will fall if you exceed the safe lean amount. You could as well use the steer and compensate for the turning leaning. this last one, being harder to control as turning amount didn't went back to 0 whenever you released turning key.

A great thing to note is that this script was aimed at MotorBikes so weight and everything had sense, and physics were accurate. a bicycle should be twitchy, less weighty, and less powerful so there was no way I implemented it this way but I took a great note about how I

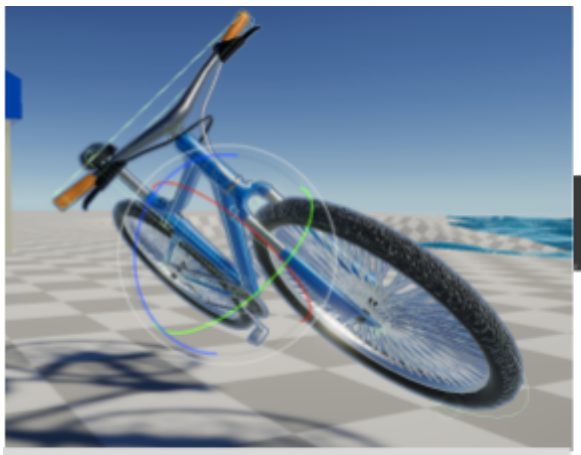
could.

I unfroze the rotation on the Z and tried the physics approach. I went on for a couple of hours trying to stabilize the bike slightly offsetting the COG that the rigid body had on the direction of the turn.



I could not get it to work, we need to find a different way around

I decided to reengage with the idea of freezing the Z axis of the rigid body and transforming the Z axis of my bike, as whenever I turned if I smoothly rotate the bike in the Z axis towards the direction of the turn, I can “emulate” the leaning of bikers with a smooth step function



The orientation//Rotation of the Z axis is the Blue Circle

Freezing teh rotation of the rigidbody on the z axis.

For this, I’ve created a float value that stores the max laying amount “+” and “-”. Whenever we turn we multiply the value for the value of our input Steering which is always 1 or -1. this just changes the orientation of the lean, to rotate towards -40 or 40.

```
1 referencia
public void HandleSteering()
{
    currentSteeringAngle = Mathf.Lerp(currentSteeringAngle, maxSteeringAngle * horizontalInput, 0.05f);
    frontLeftwheel.steerAngle = currentSteeringAngle;

    //We set the target laying angle to the + or - input value of our steering
    //We invert our input for rotating in the ocrrect axis
    targetlayingAngle = maxlayingAngle * -horizontalInput;
}
```

In the LayOnturn we basically set the values for when are we leaning and the smooth step value for that function.

```
1 referencia
private void LayOnTurn()
{
    Vector3 currentRot = transform.rotation.eulerAngles;

    if (rb.velocity.magnitude < 1)
    {
        layingammount = Mathf.LerpAngle(layingammount, 0f, 0.05f);
        transform.rotation = Quaternion.Euler(currentRot.x, currentRot.y, layingammount);
        return;
    }

    if (currentSteeringAngle < 0.5f && currentSteeringAngle > -0.5  ) //We're stright
    {
        layingammount =  Mathf.LerpAngle(layingammount, 0f, leanSmoothing);
    }
    else //We're turning
    {
        layingammount = Mathf.LerpAngle(layingammount, targetlayingAngle, leanSmoothing );
        rb.centerOfMass = new Vector3(rb.centerOfMass.x, COG.y, rb.centerOfMass.z);
    }

    transform.rotation = Quaternion.Euler(currentRot.x, currentRot.y, layingammount);
}
```

After everything we had a bicycle script that allowed us to tilt the bike in the direction of the turn. and to accelerate and brake accordingly.

To properly control the Handle we simply rotated the handle on the Y axis setting it as the current steering angle in the local axis of our HandlePivotPoint

```
1 referencia
public void UpdateHandle()
{
    Quaternion sethandleRot;
    sethandleRot = frontWheeltransform.rotation;
    handle.localRotation = Quaternion.Euler(handle.localRotation.eulerAngles.x, currentSteeringAngle, handle.localRotation.eulerAngles.z);
}
```

We established another function to control the max amount of steering angle depending on how fast you were going, if too fast we would decrease the amount of steering angle to a usable degree at that speed, This is because the bike was flipping usually even at low speeds and was too difficult to drive at high speeds

```
1 referencia
public void SpeedSteerinReductor()
{
    if (rb.velocity.magnitude < 5 ) //We set the limiting factor for the steering thus allowing how
    {
        maxSteeringAngle = Mathf.LerpAngle(maxSteeringAngle, 50, speedteercontrolTime);
    }
    if (rb.velocity.magnitude > 5 && rb.velocity.magnitude < 10 )
    {
        maxSteeringAngle = Mathf.LerpAngle(maxSteeringAngle, 30, speedteercontrolTime);
    }
    if (rb.velocity.magnitude > 10 && rb.velocity.magnitude < 15 )
    {
        maxSteeringAngle = Mathf.LerpAngle(maxSteeringAngle, 15, speedteercontrolTime);
    }
    if (rb.velocity.magnitude > 15 && rb.velocity.magnitude < 20 )
    {
        maxSteeringAngle = Mathf.LerpAngle(maxSteeringAngle,  10, speedteercontrolTime);
    }
    if (rb.velocity.magnitude > 20)
    {
        maxSteeringAngle = Mathf.LerpAngle(maxSteeringAngle,  5, speedteercontrolTime);
    }
}
```

We ended up adding an extra function for better handling and stability, In order to keep the bike from jumping off the ground as often by tiny bumps and help control it at high speeds without losing it that easily.

```
1 referencia
public void DownPresureOnSpeed()
{
    Vector3 downforce = Vector3.down;
    float downpressure;
    if (rb.velocity.magnitude > 5)
    {
        downpressure = rb.velocity.magnitude;
        rb.AddForce(downforce * downpressure, ForceMode.Force);
    }

}
```

We as well upgraded the physics engine to be updated instead of every 0.2f to a 0.15f or 0.1f in the PhysicsTimeUpdate so it refreshes the collision faster allowing for a smoother simulation

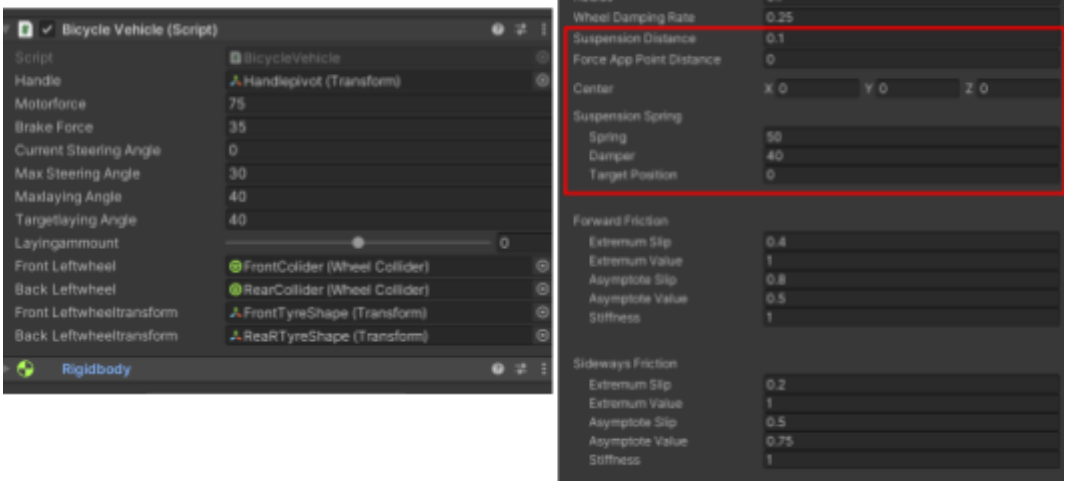
It's important to keep in mind as well, that whenever you were turning, in the LayOnTurn Function we offset the center of gravity of the rigid body downwards 0.1f thus allowing for better control of the leaning.

Then I Set off to finalize the work with a little touch to make it feel more grounded, so I added one trail Renderer on each wheel collider and attached the emitting variable to the GetGroundHit() of each wheel collider, which allows us to enable the trail every time the bike touches the ground.

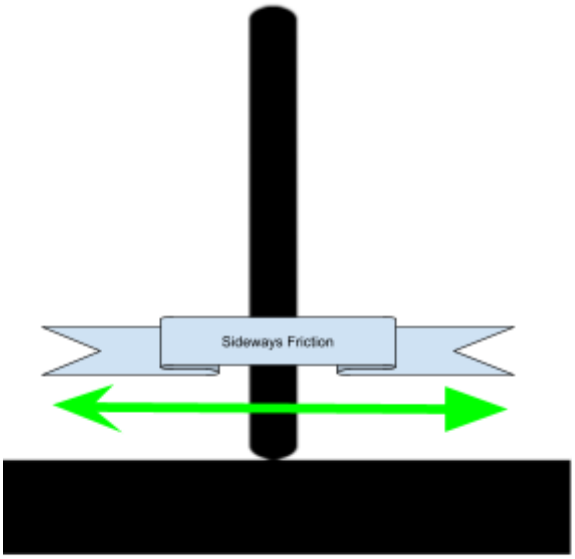
```
1 referencia
private void EmitTrail()
{
    frontGrounded = frontWheel.GetGroundHit(out WheelHit Fhit);
    rearGrounded = backWheel.GetGroundHit(out WheelHit Rhit);

    if (frontGrounded)
    {
        fronttrail.emitting = true;
    }
    else
    {
        fronttrail.emitting = false;
    }

    if (rearGrounded)
    {
        rearttrail.emitting = true;
    }
    else
    {
        rearttrail.emitting = false;
    }

    //fronttrail.emitting = true;
    //rearttrail.emitting = true;
}
1 referencia
```

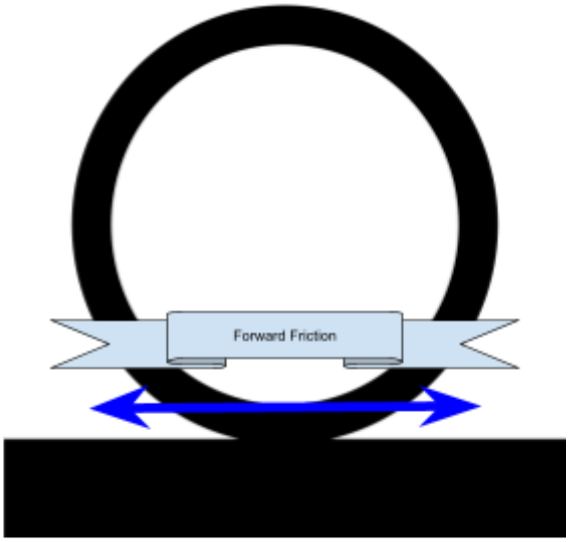
The WheelCollider's Problems

The wheel colliders had something really important to note about. you had to adjust the suspension to work with a vehicle that weighs practically nothing.

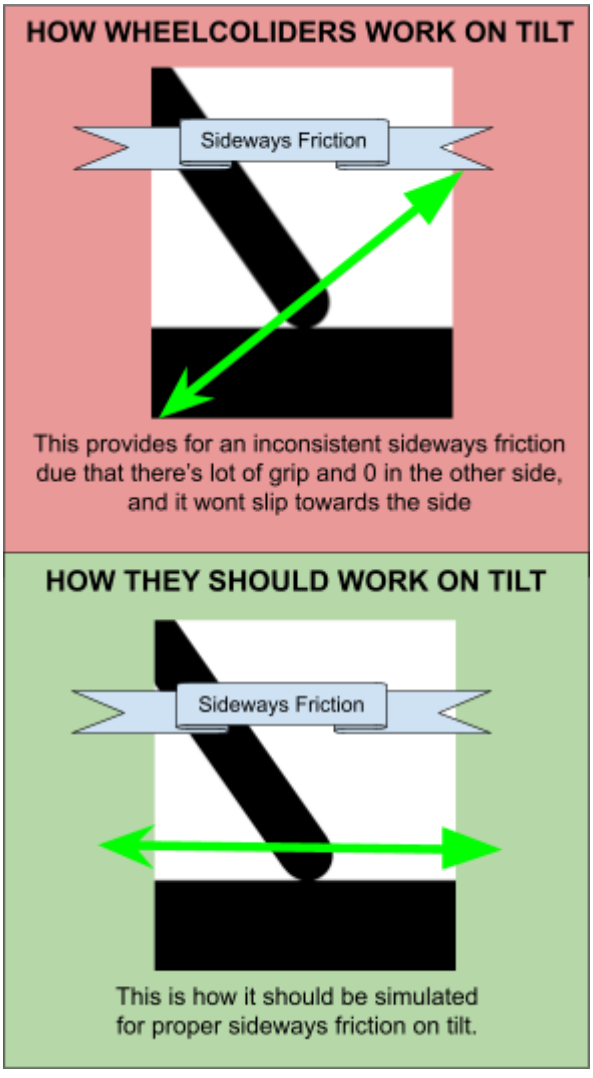


As well they're thought for vehicles with 4 wheels in mind, as the slip values are not correlated with the rotational values in the Z-axis of the wheels, only account for friction values related to the Y-axis and X-axis,





If the wheel is properly straight up, the friction values are calculated properly, and the tensor axes are used in accordance as the pictures above explain. Anything outside of this behavior, is excluded from the simulation and, actually there's no problem with this if you're making vehicles, with 4 wheels. whereas no wheel is going to tilt in any direction.

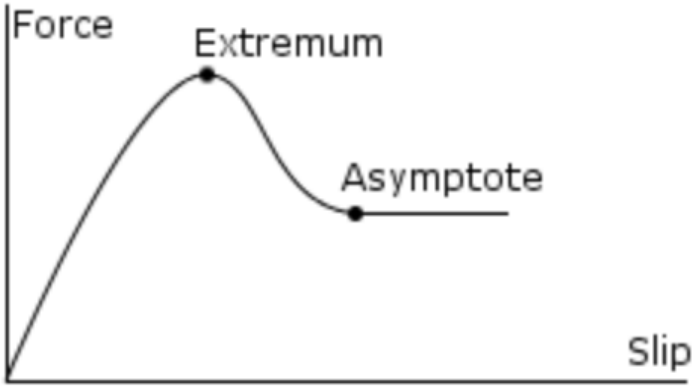


The problem with the wheel collider's friction properties is that they do not apply any friction value in relation to the tilting. Maybe the problem resides in the fact that the sideways friction should be world-dependent instead od local axis-dependent. may be needed to test.

The wheel collider as well has got only one contact point raycasting downwards, which creates several inconsistencies, when on bumpy terrain.

this plus the friction inconsistencies, leads to sometimes unexpected bike behaviors, for example when turning, fast under slow speeds they twisted fast in the turning direction due to the back wheel lifting up the ground for some reason claiming 0 grip and sending the bike on an aerial slide. that ends up with the player facing the wrong direction. I saw that similar thing in GTA 4 bikes I'm actually facing the same problem and I don't know if it's cuz of the center of gravity. or cuz of the friction values for the back tire, or a mix of both.

In order to properly set up the values for the grip levels, we had to understand the grip curve of a tire in order to replicate the behavior. Thankfully unity Documentations already have an approximate usual tire grip graph, For you to replicate and play around with.



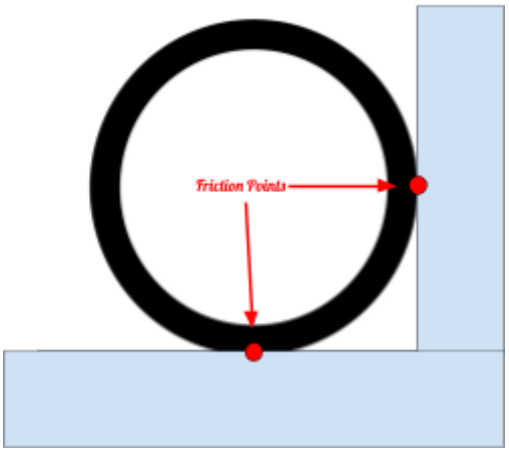
Typical shape of a wheel friction curve

The system is finished with some inconsistencies that may need to be resolved with the creation of extra assets,

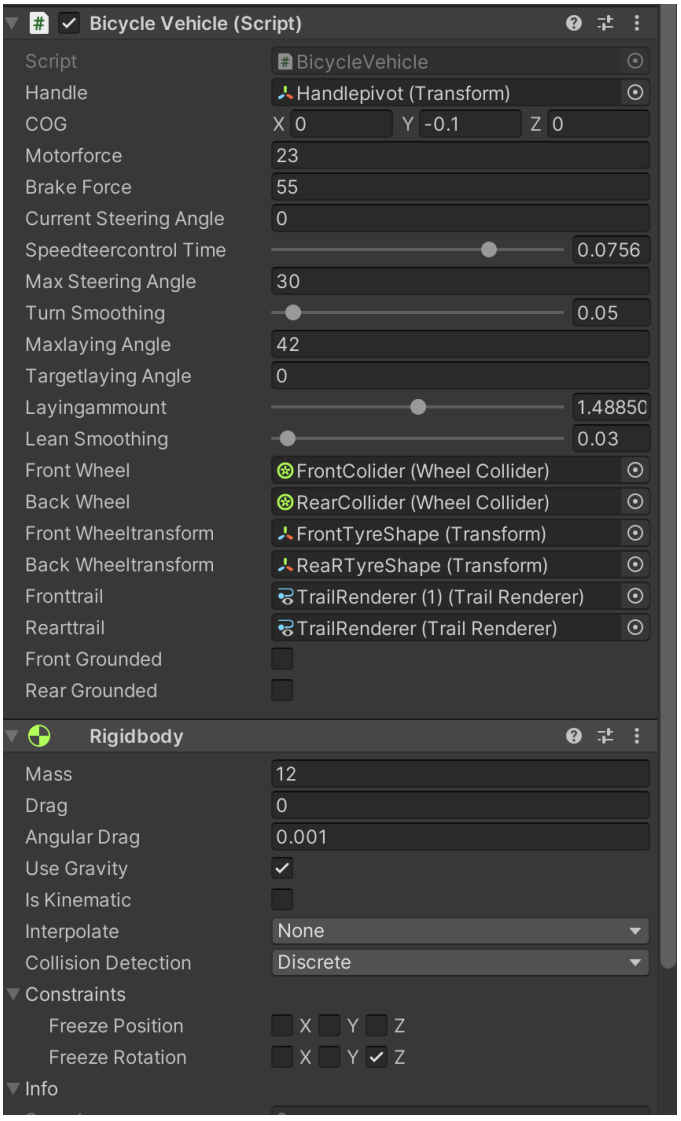
We could as well limit the tilt to a less visible tilt and allow for the tire frictions to be smoother and less janky or even remove the tilt entirely to ensure full proper friction values. but slightly lowering them until you find a sweet spot where it tilts and doesn't lose enough verticality to maintain proper friction.

And if we want the player to get extra features like tricks and stuff we can add rotation in the last 2 axes, allowing us to do Wheelies, stoppies, and 360º.

We still need to fix the issue with the wheel collider and create some Custom colliders using the sphere cast function. The reason why sphere casting may be the way is that we may be able to stipulate multiple friction points of contact even on surfaces where the wheel may friction from the down point but the front point as well



But by now the bike drives. and turns properly tilting accordingly.
I'm going to leave it here and release the asset for free. I'm probably going to keep on working on them. but at least everyone can play with it.



Thanks for reading.- Sincerely