Welcome to Bicycle PRO kit ver 1.0 !

First of all let me **thank you** for buying this Bicycle PRO Kit.

So, there I try to explain **how it works**:

First you need to know It's not real physics simulator. There is many reasons why. One of them is my first attempt to make anything like real world and absolutely not stable motorcycle(yes, it was motorcycle) which falls every time as result. It has really round wheel and ruled by principles of counter steering but no fun at all. Why ? Because of physX and wheelCollider isn't represents real world physics. It's a code, and only a fake of some principles.

Do you need physically correct bicycle which makes you angry ? I don't think so - that's why you have bought that kit. Because demo you saw is fine, smooth and fun.

So, main idea is make looking good bicycle with fancy controls.

And here we are.

How it works?

With that kit you've downloaded:
1. Four scenes with **two different bicycles models**
1. A **mecanim character with ragdoll**
2. **Sound** for skidding
3. **bicycle_code.js** - script
4. **pedalControls.js** - script
5. **bike_sound.js** - script
6. **biker_logic_mecanim.js** - script
7. **rear_suspensionAmmo.cs** - script
8. **keyboardControls.cs** - script
9. **mobileControls.cs** - script
10. **controlHub.cs** - script
11. **camSwitcher.cs** - script
12. **skidMarks.js** - script
13. **skidMarksDestroy.js** - script
14. **startScript.cs** - script
15. **toMainMenu.cs** - script

Let me explain everything in this list:

1. **Two bicycles** - there is simple models of two bicycles of different types. Just for example and understanding how same code work with different settings. Each one contains body, wheelbar with upper front forge, front forge lower part, and two wheels(red bicycle has also pendulum and rear amortizator).
   Four scenes is:
   a. BPK_MainMenu - just lobby for bicycle and type of controls selection
   b. BPK_bicycle_MTB - riding scene with dirt bicycle
   c. BPK_bicycle_FullSuspension - riding scene with full suspension bicycle
   d. BPK_bicycle_mobile - riding scene with dirt bicycle with mobile controls

1. **The mecanim character.** It's animated character, mecanim controller, mask for leg(when stopped) and ragdoll.
2. **Skid sound** for bicycle. Royalty free sounds so quality isn't best).
3. **bicycle_code.js** - main script for bicycle itself. There is everything about ride and leaning. For a ride you need only this one.
4. **pedalControls.js** - second script for simulate nature of bicycle behavior. It script works with pedals, simulates light leaning when rider is accelerating and also, contains some stunts for example..
5. **bike_sound.js** - this script controls sounds.
6. **biker_logic_mecanim.js** - script for a rider. It's controls his animations, head movement, ragdoll launch and so on. To make rider live you need this script.
7. **rear_suspensionAmmo.cs** script for rear amortizator and spring(for full suspension bicycles only)
8. **keyboardControls.cs**- script which examine keyboard for pressed keys
9. **mobileControls.cs**- script which examine mobile device's screen for touches
10. **controlHub.cs**- script which contains all controls variables

    (Next scripts is not necessary. It's only visual bonus)
11. **camSwitcher.cs** - script for changing camera. From behind to move around.
12. **skidMarks.js** - this scripts makes a skidmarks on ground when you braking.
13. **skidMarksDestroy.js** - very small script attached to every skidmark to destroy it after some time.
14. **startScript.cs** - small script for scene selection. It's not about bicycles at all :)
15. **toMainMenu.cs** - rough script for use "Escape" key to back to first scene. As script above it's not about bicycles at all.

Let's look a little closer

1. The Riding scene(most complex part of manual)

This bicycle is just a **example** how your own bicycle(made or bought) should be **divided** to parts for best visual. So, it means there is only **graphics**. No matter what model and size of parts there is. When you want to export your own bicycle from 3d modeling package just **follow** the those **principles of cutting** model: **body, two parts of front forge, two wheels**(for full suspension bicycles you need rear **pendulum** and rear **amortizator**)**.** Take care how front forge is made - it's exported NOT ANGLED but strongly vertical and it's already contain lower part(with brake). You may add ANY details you want. Also, take a look to some coll_bike_body2 objects in model. There is gameObjects made in Unity just to make collider boundaries. You might make it anything you like. It's for collision and funny rotation bicycle after fall.

The main script **bicycle_code.js**

Inside the code every variable have a nice name which explains point of this variable or comment which explains how this variable is used.

To make work your bicycle with that code you should take **your models(body, front forge, two wheels)** and construct the **same tree** as on the picture:
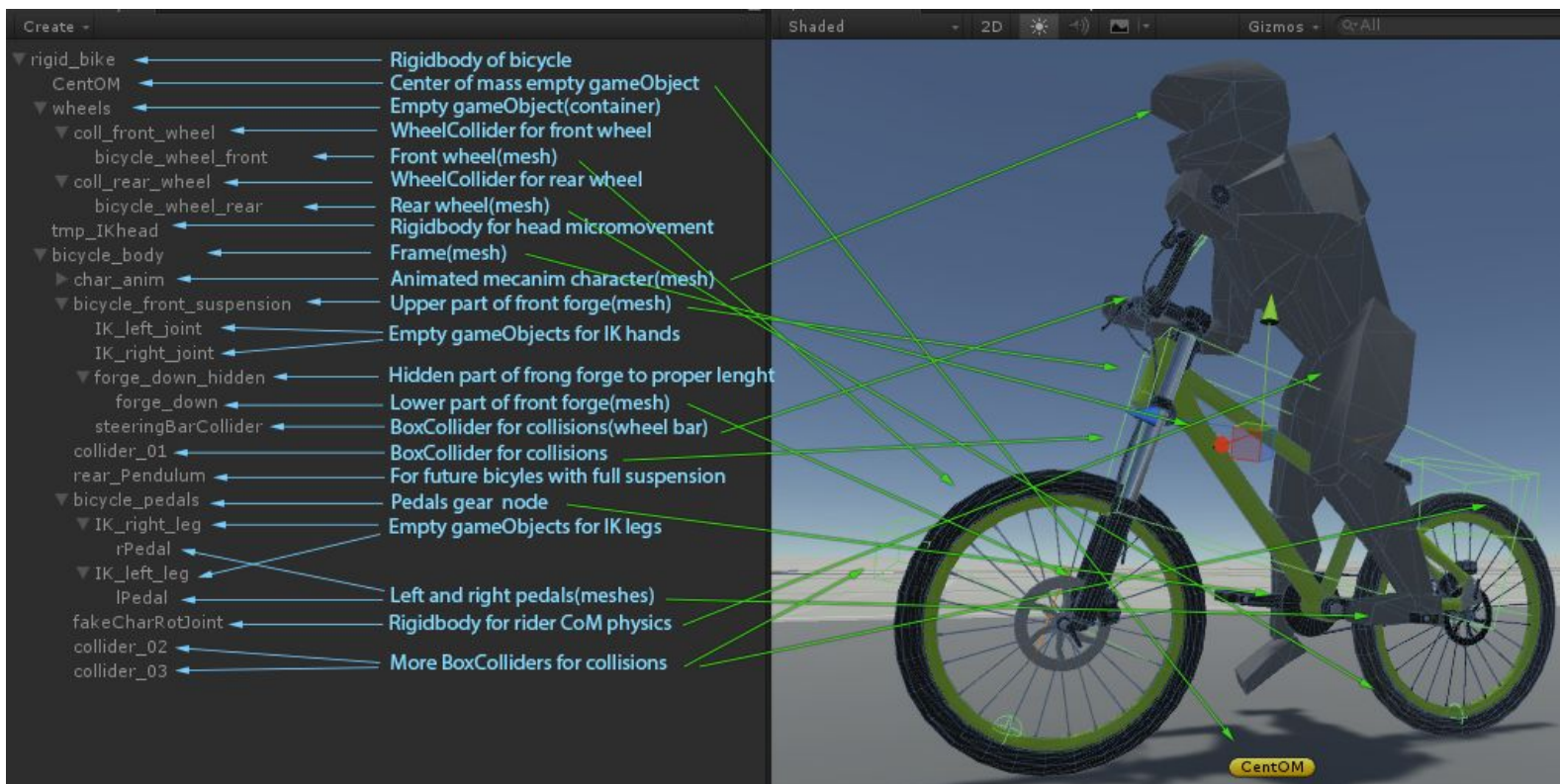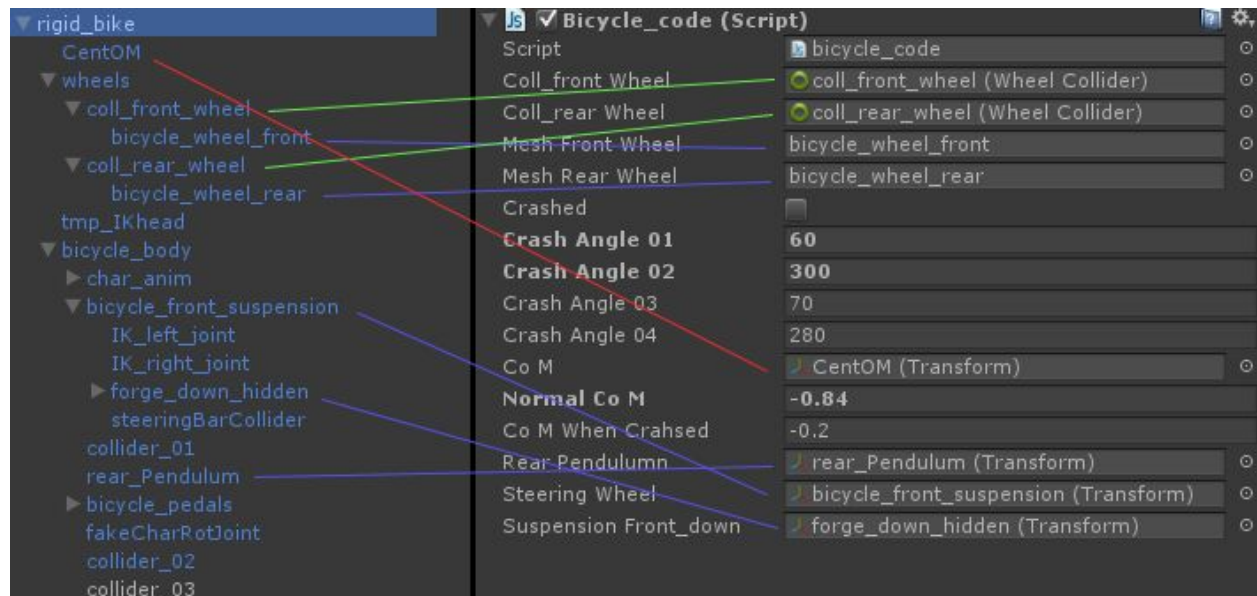
rigid_bike settings:

First of all, make sure to **change weight** of rigid body to near 200kg. Sorry for that "Magic numbers" but real weight(like 15kg) will give you awful results with new PhysX 3.3. Everything else(drag and angular drag) is doesn't matter because it's controlled by script.

Make a connections like it show below:

Blue arrows for geometry(meshes)
Green arrows for wheelColliders
Red arrows for Transform

Coll_front Wheel - this you should link to front wheel collider gameobject
Coll_rear Wheel - this you should link to rear wheel collider gameobject
Mesh Front Wheel - link it with front wheel mesh(FBX)
Mesh Rear Wheel - link it with rear wheel mesh(FBX)
Co M - link it to CentOM gameobject. After created you created your own bicycle tree, place Center of Mass empty gameobject just few centimeters **above** the line where wheel touches ground.
Steering Wheel - link to wheel_vis(FBX). So, it's wheelbar of front forge.
Rear Pendulum - link to rear_pendulum(FBX or empty GameObject). This is for bicycles with full suspension. In case of this dirt bike there is no rear suspension.
RearSusp_ammo(only full suspensiob bicycle) - amortizator(FBX) in two parts: ammo itself and a spring.
Suspension Front_down - link to supp_down_vis mesh(FBX). It's lowest part of front forge which looks at front wheel mesh and falls down when it's a gap below wheel.

Time to explain some magic numbers :)

Crash Angle 01	60
Crash Angle 02	300
Crash Angle 03	70
Crash Angle 04	280
Co M	CentOM (Transform)
Normal Co M	-0.84
Co M When Crahsed	-0.2
Rear Pendulumn	rear_Pendulum (Transform)
Steering Wheel	bicycle_front_suspension (Transform)
Suspension Front_down	forge_down_hidden (Transform)
Wheelbar Restrict Curve	
Front Brake Power	25
Legs Power	16
Air Res	6

Crash Angle 01-04: it's angles when bicycle takes "crashed" status - too much lean. 01-02 is for right/left angle. 03-04 is for front(stoppie)/rear(wheelie) angle.

Normal Co M(normal Center of mass): it's CoM Y coordinate. You need experiments to find fine CoM for your own bicycle.

Co M When Crashed: you need put up CoM when bicycle crashes. Just for funny turning when crashed.

Front Brake Power: this means very good SRAM brakes for bicycle.

**Legs Power:** power of rider's legs :) 16 means he can ride approx 45 km/h

Air Res: it's for air resistance at high speed. 1 is neutral. Make more for really big bicycles and smaller for children bicycles or sport bicycles with very good aerodynamics.

Now, last trick in that script.
In real life controls of bicycle at slow speeds and high speeds a little bit different. That calls "countersteering". This is physics of real life. Here, to achieve this effect and make bicycle more stable on high speed we need to clamp wheelbar angle according the speed. It means - the faster bicycle rides the less angle you can rotate wheel bar.
This is example curve for demo dirt bicycle.
Y(vertical) - wheelbar angle, X(horizontal) - bicycle speed
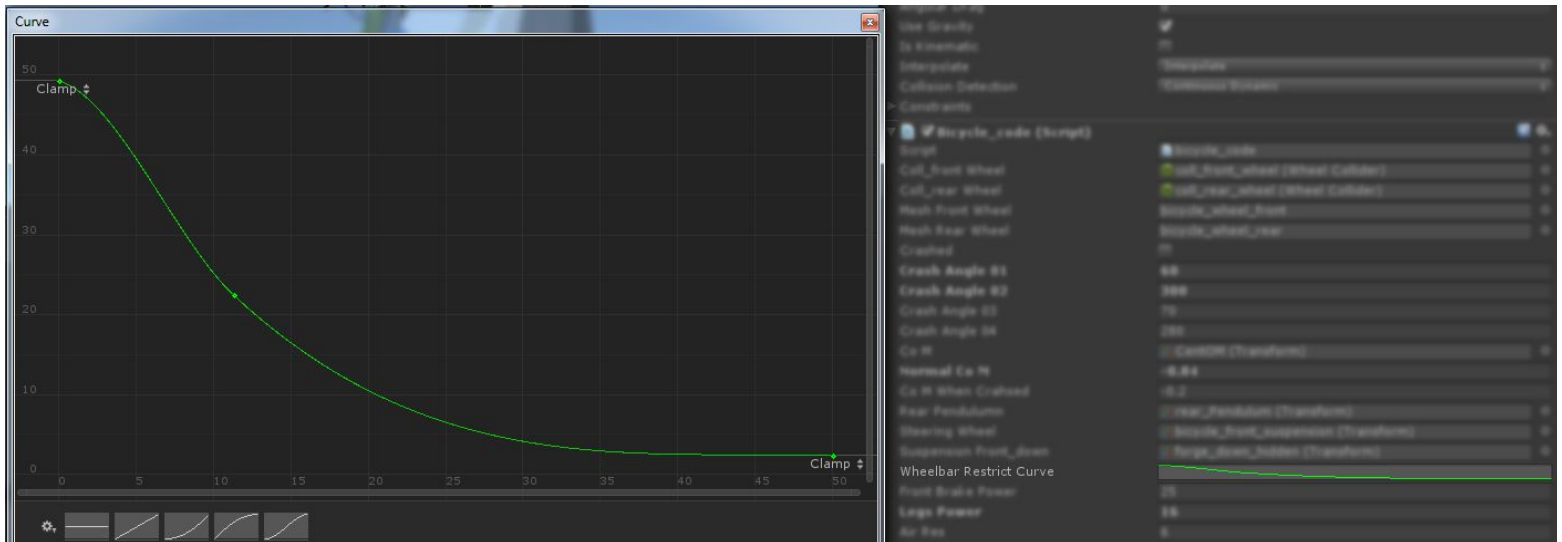
For your bicycle tune this curve manually in Editor to get something like that:
at speed 0 km/h = 49 degrees of wheel turning
at speed 10 km/h = 25 degrees of wheel turning
at speed 20 km/h = 10 degrees of wheel turning
at speed 40 km/h = 3 degrees of wheel turning

at speed over 40 km/h - do not need to be modify because this is pretty speed and there is no reason to make cornering radius lesser
Curve ends at 50km/h so, last meaning applies to speed over 50km/h(crazy downhills)

Main rule for this tuning is - "the more angle is more dangerous" :) so, make wide turning angles and slow speeds and make it very tight at high.

And last thing you should know is - **wheelCollider** settings.
This is good **wheelCollider settings** for front wheel of dirt bike(play with settings for other kids of your bicycles):
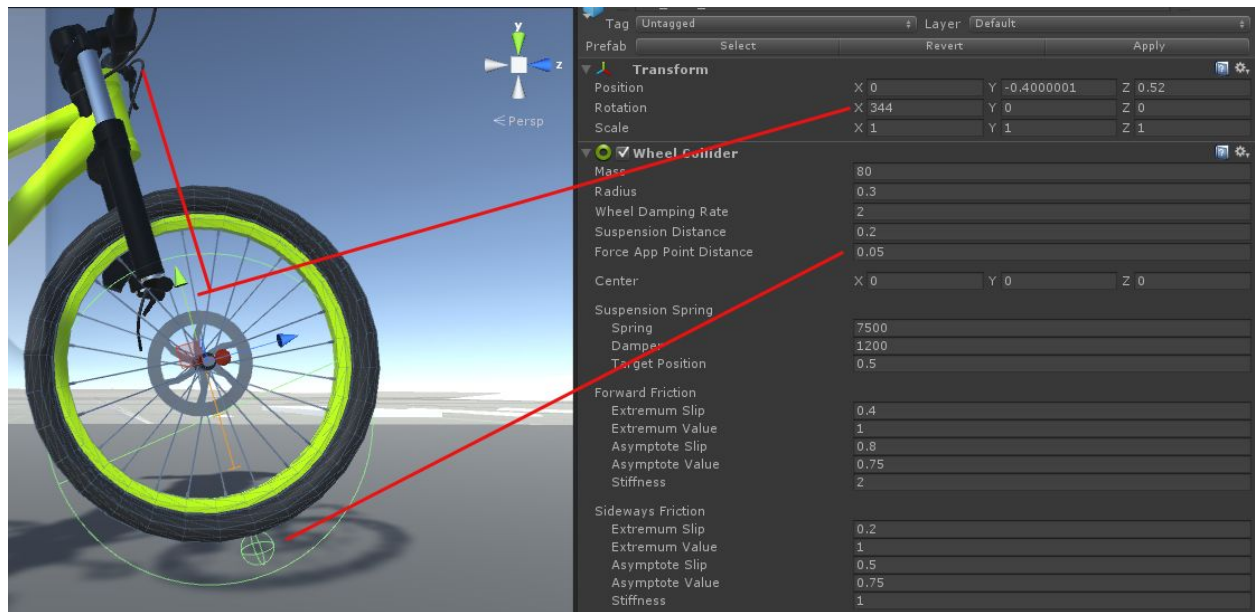
Mass 80kg for a wheel is another new PhysX 3 "requirements".
Pay attention for **Rotation X 334**. It's 26(360-334 = 26)degrees for front forge angle. The front forge **takes this angle by script.**
Next important thing is **Force App Point Distance** is 0.25.
It's ONLY for front wheel. The rear one should have 0 at this option.
Copy all settings showed on picture for a start. Then try to find good **Spring** and **Damper** for your bicycle.

For a **rear** wheelCollider settings is same, except:
Suspension Distance - 0.2 and **Force App Point Distance**. It's **0(zero)**.

The second script for bicycle is **pedalControls.js**
This script account for stunt's and pedal rotation.
Also, it simulates left/right small leanings when rider pressing pedals.
Here is connection of nodes of **pedalControls.js** to **bicycle_pedals** gameObject:
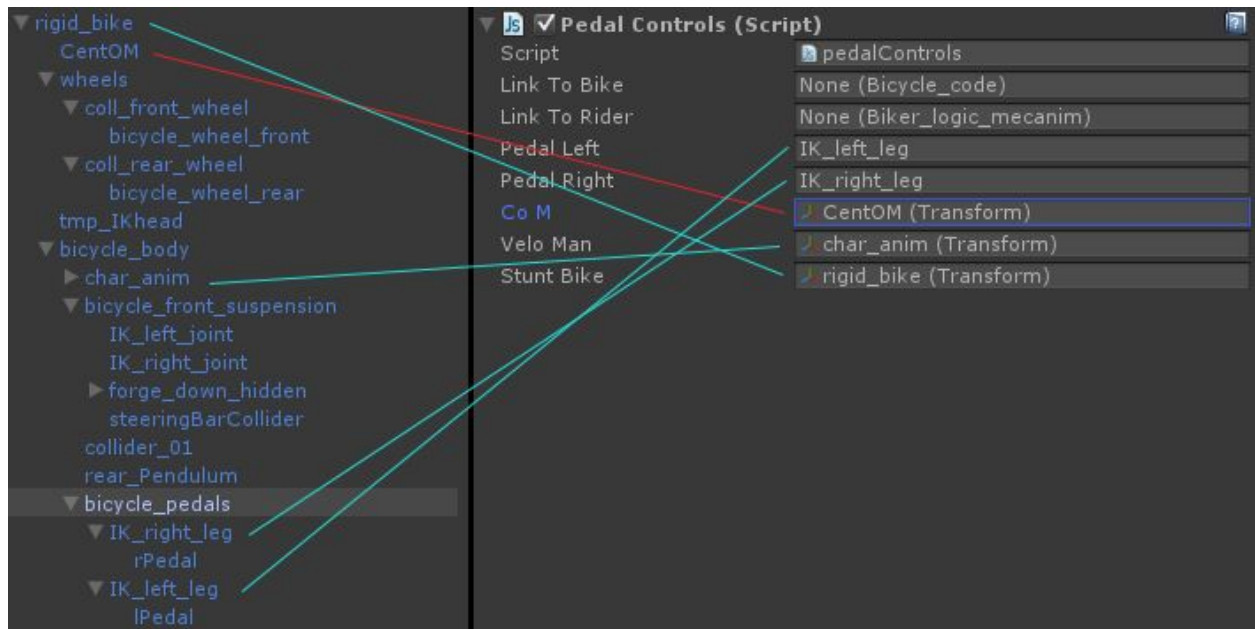
That's all you need to know about bicycle settings. Other settings(not necessary, like sounds and skidmarks) will be explain later.

2. The mecanim characters(not so complex but long part of manual)

In that section I'll explain how to set up character as a rider.

You need **2.fbx** files:

      a. **char_mechanim.fbx**

      b. **char_anim.fbx**

There is meshes for rider. First - char_mechanim.fbx is just mesh for mecanim setup. Next one is animated mesh(char_anim.fbx)

The char_mechanim.fbx is mapped as usual mecanim avatar from mecanim manual:
http://docs.unity3d.com/Manual/ConfiguringtheAvatar.html
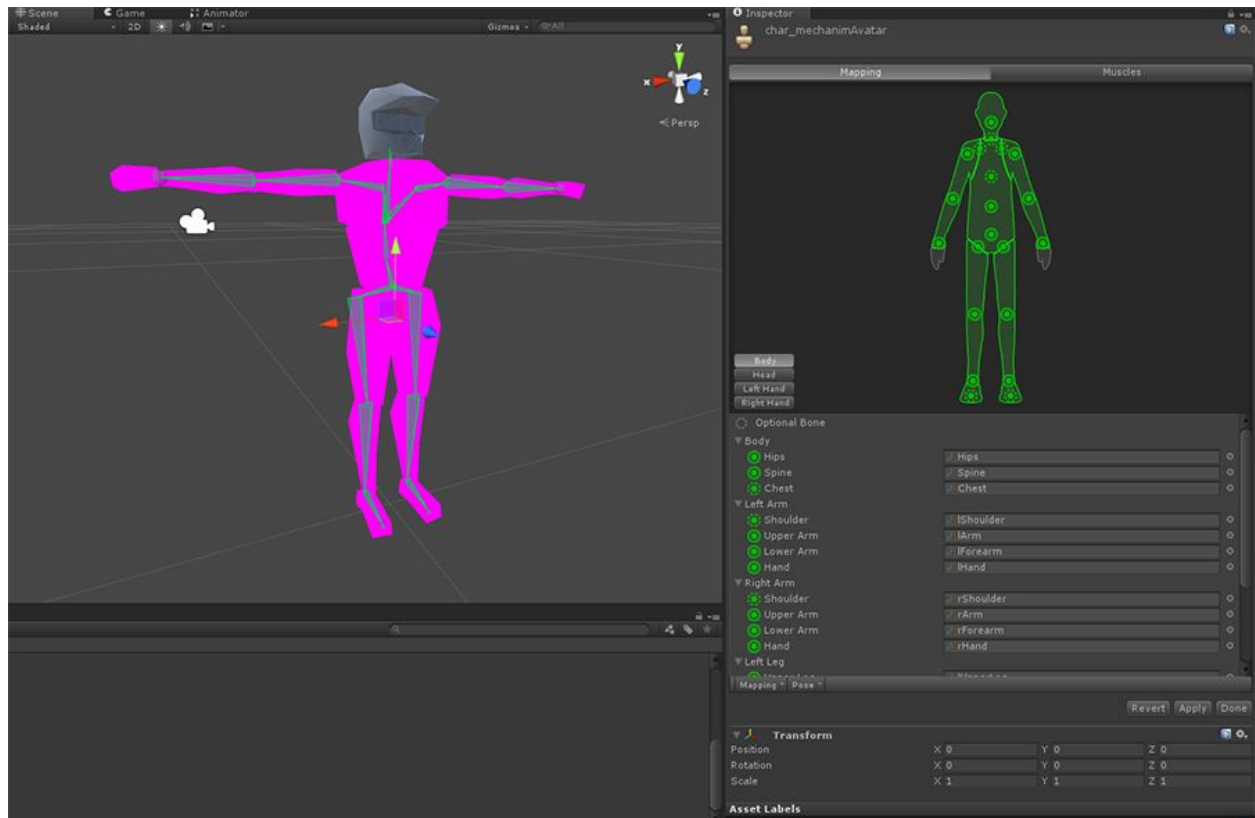
All we need is map bones correctly. We need that for add any kind of riders after.

To save our time to add bikers with different poses, sizes and so on. It will be our main instance for bones mapping.

No animation on this character, just select "Humanoid" and "Create from This Model"
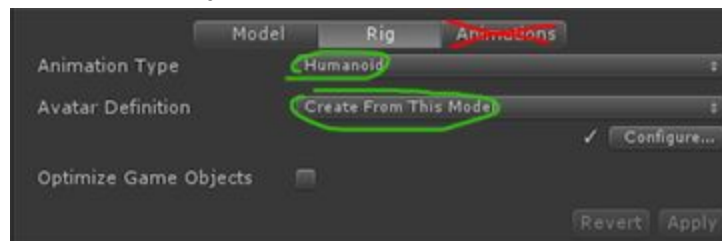
Now, time for **char_anim.fbx**

First of all you need to make animations in 3d package for your character or get it any other way(download for example).

I've used 6 animations: idle, RighLean, Leftlean, moveForw, moveBack, legOff. You might use any names and any number of animations. The names says about themselves. legOff is animation when rider put down his leg(s) to ground when bicycle not moving.

Next step is set up new character.fbx.

First step is choose Animation Type - "Humanoid", then choose "Copy From Other Avatar" for Avatar Definition, and choose your created before mecanim avatar with properly mapped bones as Source.

Now your new character rider perfectly mapped as instance avatar and you need no more to do that for any new character.

Then, go to section "Animations" and create animations with timing from your 3d file(I've used all animations in one file, you may keep animations in different files)
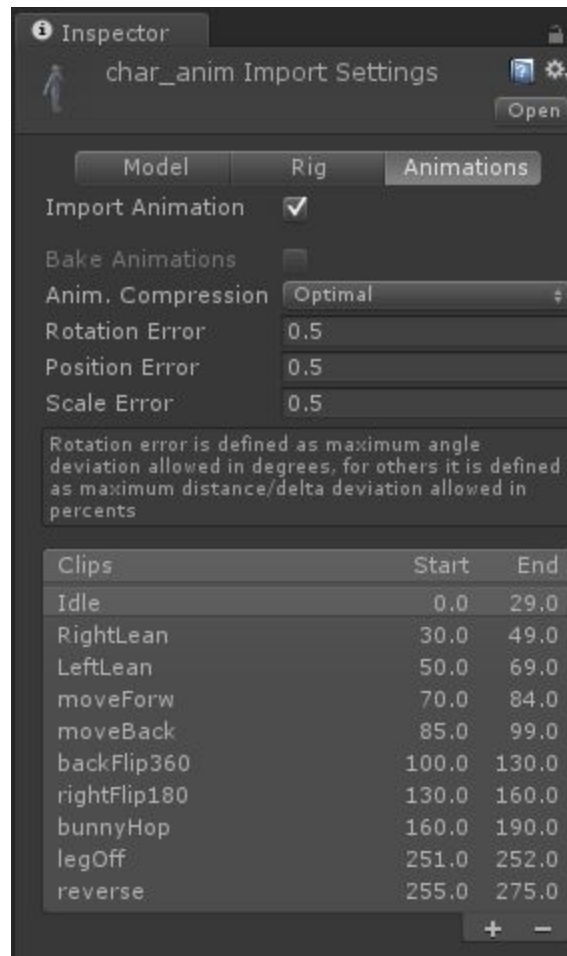
Do NOT forget to check **"Loop Time"** for **idle** animation :)

Stunts animation like backFlip360, rightFlip180, bunnyHop is optional.
Animation names explain what it doing.
As for legOff - it's animation with right leg is standing on ground for zero speed idle.

Your character is ready to play animations and ride a bicycle. To play it well we need few steps more. To achieve this we need to set up animator and then link some variables in script **biker_logic_mecanim.js**

1. Drag your character.fbx to rigid_bike/body(as you read before in chapter "The Riding Scene").
2. Create animation controller for mecanim: Assets/Create/Animation Controller and name it anything you like. For example "myNewCharController".
3. Go to Animator window(Window/Animator). Right click in a middle of window /Create State/Blend Tree
4. Select "2d Simple Directional" for Blend Type.
5. Add your animations by "Add motion Field"
6. Make it same as on the picture below
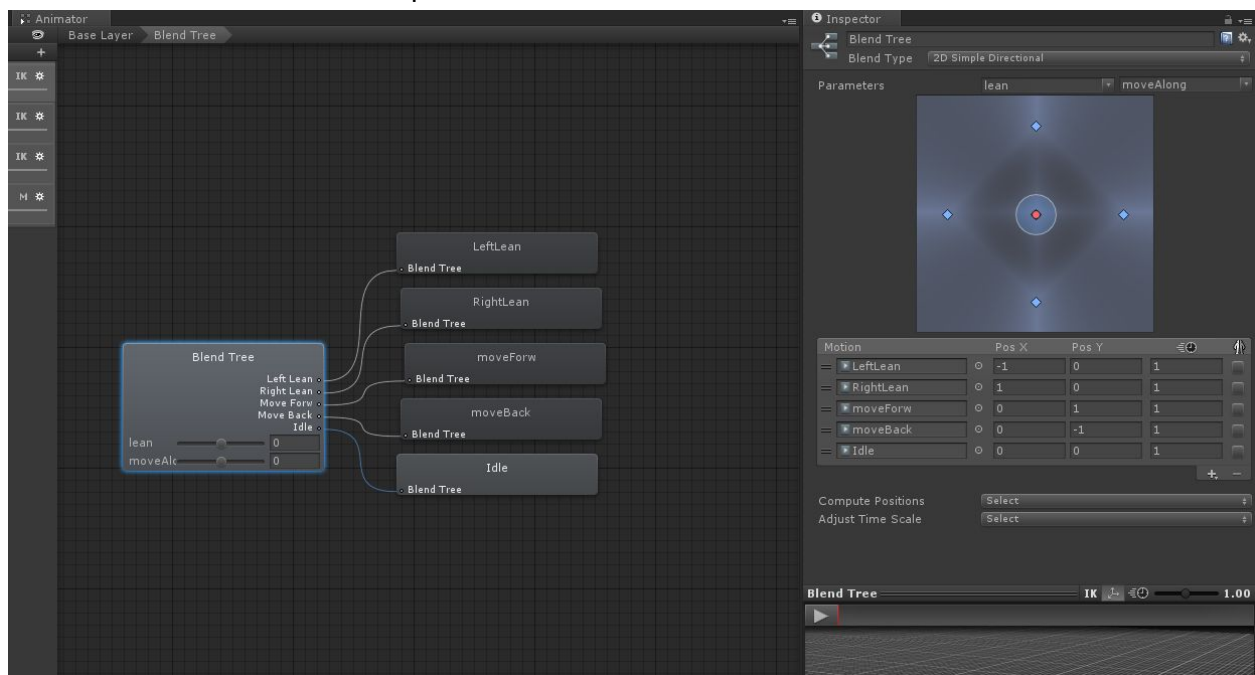
idle is in middle, and other ones is like it sounds - **RightLean** at right, **moveForw** at up side and so on.

So, it means when rider do nothing he playing "idle". When he does some movies "**biker_logic_mecanim.js**" plays it and this "blender" mixes it with idle ! Really cool.

Ok, what's next ?

6. We need to make layer mask to exclude all body in animation of put leg down to the ground(legOff). To achieve this, you need Assets/Create/Avatar Mask and make it like on the picture:

Just one leg is green(every other bones will play their own animations but leg will play "to the ground").

7.Then we need create new animation layer in Animator window for our controller "myNewCharController". But before make current layer(higher) IK(click on gear and check "ik pass"). Then create another layer, name it "legOff" or something like that. Click on gear at new "legOff" layer and select mask you created before.
(I know it's too much action to make it with no mistake but you have my controller **"RiderAnimController"** as an example).

8. In that "legOff" layer Righ Click/Create State/From New Blend Tree. Then create "1D" Tree type and link to your put leg to ground animation. That's it !

So, back to new character.fbx you have put to rigid_bike/body
Add Component "Miscellaneous/Animator" to this character.
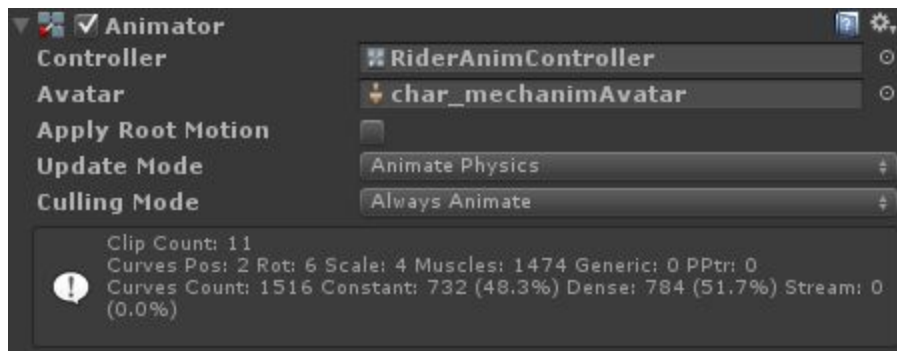
Choose your Controller which you've made in step 2(myNewCharController)
Choose Avatar - the instance one which you made at very beginning of this chapter.

We need last thing - add script **biker_logic_mecanim.js** to new character and link some variables as it shown on picture below:
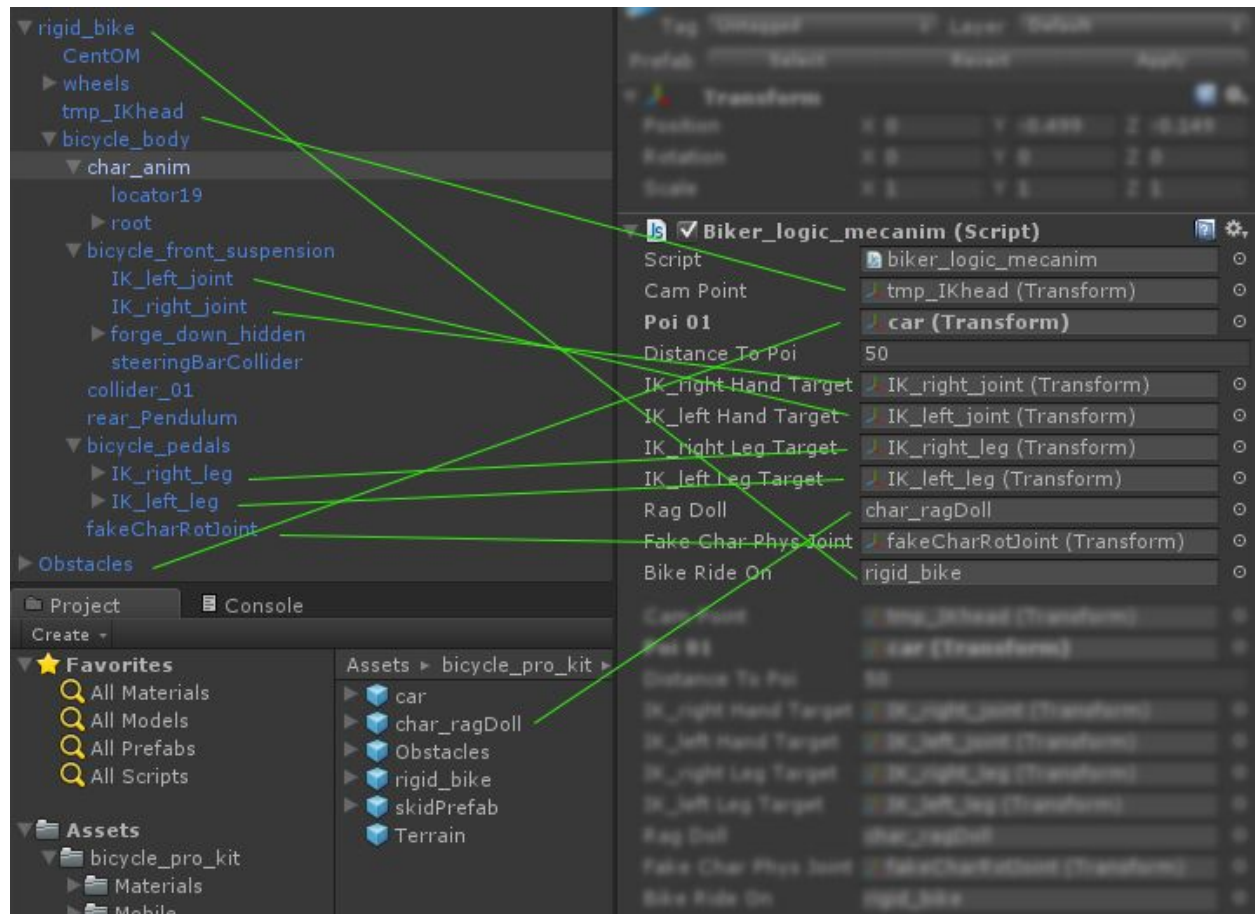
Cam Point link to tmp_IKhead in rigid_bike. It's a little rigidbody with small mass to make head movement little jitter and some inertia follow.

Poi 01 link to any object of interest in the scene. The rider will look at on that object(sure, you can make many Points of Interests).

Distance to Poi in meters. There is distance in meters when rider should look at Poi.

IK_right Hand Target link to empty gameObject at **right** side of wheelbar in rigid_bike. You need to follow arm to wheelbar.

IK_left Hand Target link to empty gameObject at **left** side of wheelbar in rigid_bike. You need to follow arm to wheelbar.

Rag Doll link to ragdoll prefab in Prefabs folder.

Fake Char Phys Joint link to fakeCharRotJoint in rigid_bike. It's a little rigidbody with small mass to make fancy rider moves to simulate inertia.

Ride Bike On link to rigid_bike itself.

That's ALL ! No more hard things in that manual ! I promise.

3. Sound for skidding

All sounds works by **bike_sound.js** script. Skidding when brake is pressed.

Just want to say again - this sound is royalty free and quite bad quality. Find better one and your bicycle will buuuuurn ears !

4. bycicle_code.js

There is main script bicycle physics and movement. See detailed info in comments in script.

By that script you may control any bicycle with no changes in script. Everything you need is different geometry, nice Center of mass placement and reasonable presets.

Just put this script to your rigid_bike and link variables as it shown at chapter 1.

5. pedalControls.js

There is second script for bicycle's behavior. See detailed info in comments in script. Also, pay attention to stunts examples in that script. If you want to make your own stunt, follow the way how other stunts are made.

Just put this script to your rigid_bike/bicycle_pedals and link variables as it shown at chapter 1.
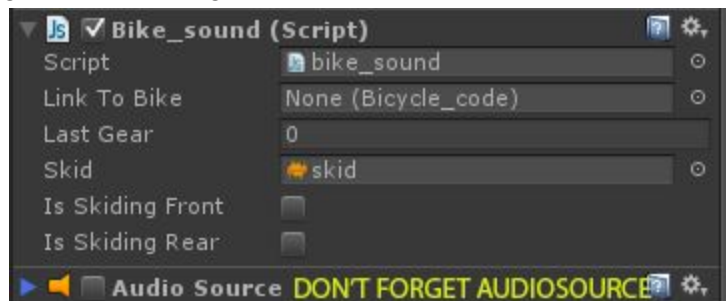
6. bike_sound.js

You should add AudioSource to your bicycle.

Put this script on rigid_bike.

Then link sounds you want to these variables:

Skid for skidding sound. Nothing to explain.

7. biker_logic_mecanim.js

This controls the rider behavior.

Put this script on your character and link variables as in explained in chapter 2.

8. rear_suspensionAmmo.cs

Use this one only when you got bicycle with full suspension(rear pendulum). This script aims rear amortizator to target(invisible gameObject) of rear pendulum and squeezes the amortizator spring when need it.

Just connect nodes as shown on picture below:
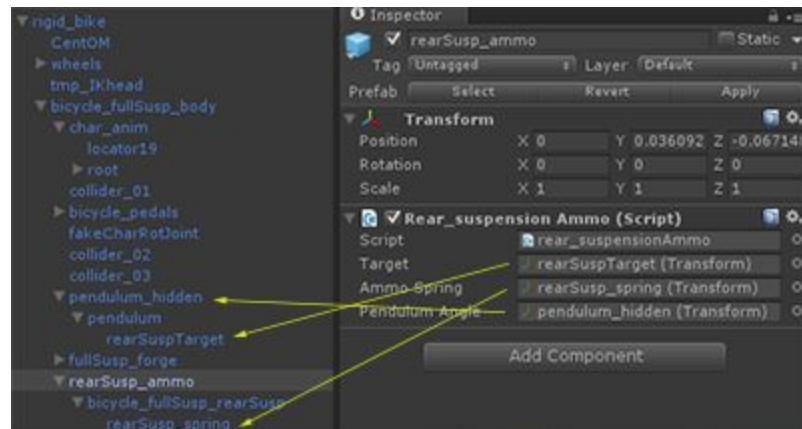
So, you can build ANY kind of two suspension bicycle. Rear amortizator will always will look at your rear pendulum joint.

9. keyboardControls.cs

This script pings keyboard for any key pressed. If this happens, it's translate this to **controlHub.cs** - another script which contains all variables for bicycle controls. This complex scheme made for comfortable change of controller type.

Put this script to empty gameObject called "gameScenario":
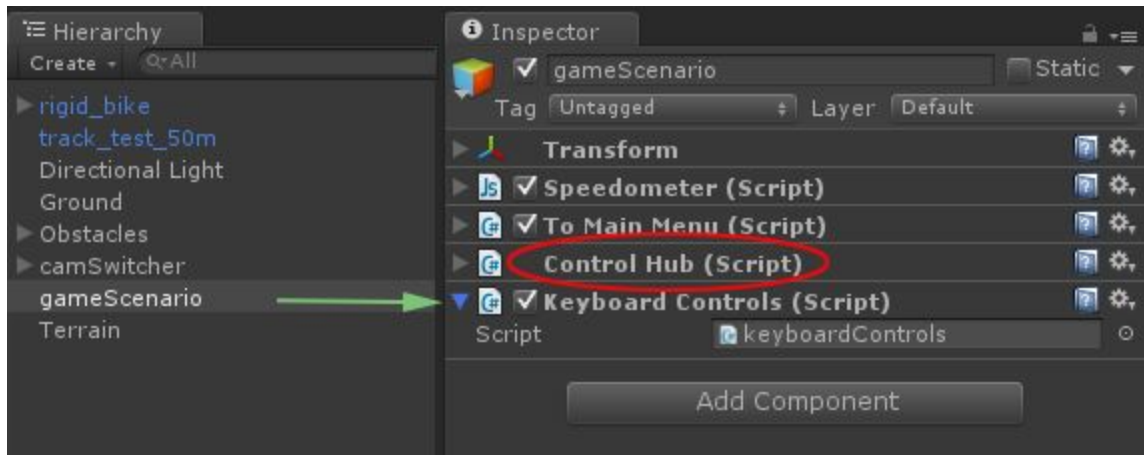
Make sure the "gameScenario" contains script "**controlHub.cs**" as well.

10. mobileControls.cs

This script pings mobile device's touch screen for any touches. If this happens, it's translate this to **controlHub.cs** - another script which contains all variables for bicycle controls. This complex scheme made for comfortable change of controller type.

Put this script in mobileControls(GameObject/UI/Canvas) canvas type UI element:
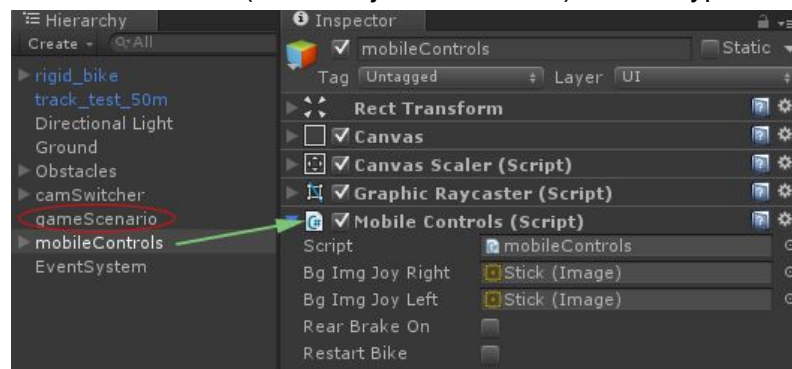
Make sure the "gameScenario" contains script "**controlHub.cs**".

11. controlHub.cs

This script only contains empty variables which ones can be read by other scripts(**bicycle_code.js** for example). These variables may be changed from any other scripts. The sequence looks like: controlsScripts->**controlHub.cs-**>execution scripts(bicycle, rider, sound and so on).

Why so hard ? You might ask.

This made for easy changing type of controls. All you need is write just one control script to make your bicycle controled by any device: joystick, leap motion and so on.

Keyboard and mobile controls already done and including in this kit.

Pay attention to folder where this script **MUST** be - it's **folder "Plugins"** in main directory of project. It's made for situation when Java Scripts looking for variables in C# scripts. Placing C# script to folder "Plugins" is the only way to compile it before JS scripts. We need this for read from this C# in any other programming languages scripts.

12. camSwitcher.cs

Simple script for camera controls. There is two cameras in scene. First one is "behind" camera turned on by default. Second os "rotate around" camera. Activated by RMC(right mouse click). When activated you may look at your motorcycle from any side and zoom by the mouse wheel.

Also, there is some options in script to make camera more "action" like dynamic FoV(field of View).

There is empty gameObject named "**camSwitcher**" which include both cameras.

The script is on that object.

Back Camera - connect in Editor to your first camera

**Around Camera** - connect to your second camera

Camera Target - connected to bike bodyframe(any part of motorcycle or rider you want)
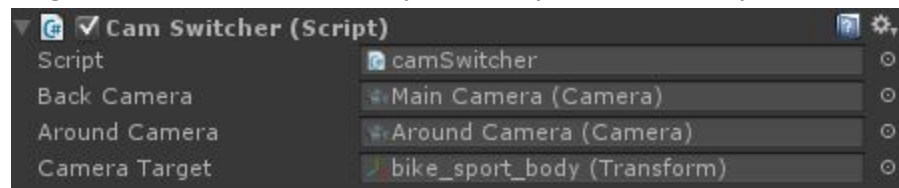
13. skidMarks.js

This script generates skidmarks when bicycle braking hard or drifting.

Just put this script on rigid_bike and connect **Skid Mark Decal** to skidPrefab in Prefabs folder. You may not linking **Link to Bike** to anything. It's automatic connects to bicycle script.
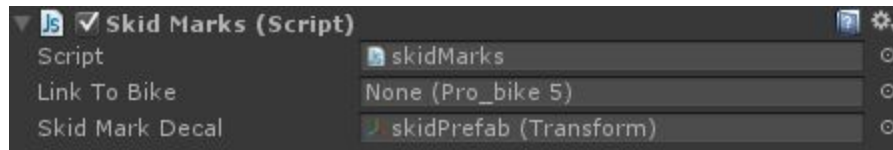

14. skidMarksDestroy.js

Very simple script already connected to skidPrefab. You need this to destroy every skidmark in scene in defined time.

Destroy This Time for time in seconds when the mark will be destroyed from scene.

Alpha Fade(abstract - not in seconds or so) for time when opacity skidmark should disappear(become fully invisible). You need this for smooth fade out skidmarks before destroy it.
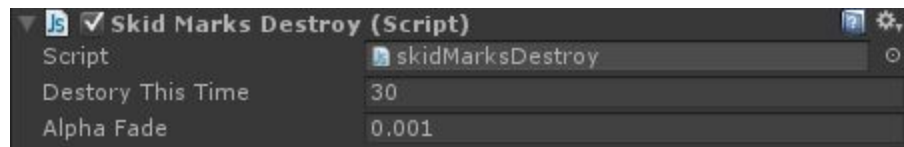
15. startScript.cs

This script is for scene **"BPK_MainMenu"**. It's used only for selection of bicycle you want to ride. Nothing interesting to explain.

16. toMainMenu.cs

This script is on empty gameObject called "gameScenario". Just pings "Escape" key to make back to **"BPK_MainMenu"** for bicycle selection. Delete it for sure if you don't need it. Nothing interesting to explain.

That's all.

Feel free to ask me anything about that kit by e-mail: **smokerr@mail.ru**
Boris Chuprin 2016