Welcome to sportbike pro kit ver 1.6 !

new features marked by blue fonts

First of all let me **thank you** for buying this sportbike 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 which falls every time as result. Yes, 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 bike 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 motorcycle with fancy controls.

And here we are.

How it works?

With that pro kit you've downloaded:

1. Five scenes with **three different motorcycle models**
2. A **three mecanim characters with ragdoll**
3. Few **sounds** for both bikes
4. **BikeController.cs** - script(1.6 fixes)
5. **BikeAudio.cs** - script(1.6 fixes)
6. **BikerAnim.cs** - script(1.6 fixes)
7. **keyboardControls.cs** - script
8. **mobileControls.cs** - script
9. **controlHub.cs** - script
10. **camSwitcher.cs** - script
11. **BikeSkid.cs** - script(1.6 fixes)
12. **SkidMarksDestroy.cs** - script(1.6 fixes)
13. **SpeedoMeter.cs** - script(1.6 fixes)
14. **startScript.cs** - script
15. **toMainMenu.cs** - script

Let me explain everything in this list:

1. **Three bikes** - there is simple models of three motorcycles 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, rear pendulum and two wheels.
   Five scenes is:
      a. u5bike_MainMenu - just lobby for bike type selection
      b. u5bike_JS - scene with sportbike and settings for this bike type
      c. u5bike_JS_chop - scene with chopper and settings for that bike type
      d. u5bike_JS_motard - scene with motocross and settings for that bike type
      e. u5bike_JS_mobile(folder **Mobile**) - scene with sportbike and mobile controls

2. **Three mecanim characters.** It's three animated characters, mecanim controllers, masks for legs(when stopped) and ragdoll. Ragdoll is same for all character.
3. Few **sounds** for bike. Engine sounds, tyres, gearbox and so on(royalty free sounds so quality isn't best). Some of them is records of car engines not bikes, so change it to proper.
4. **BikeController.cs** - main script for bike itself. There is everything about ride and leaning. For a ride you need only this one.
5. **BikeAudio.cs** - this script controls all sounds.
6. **BikeAnim.cs** - 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. **keyboardControls.cs** - script which examine keyboard for pressed keys
8. **mobileControls.cs** - script which examine mobile device's screen for touches
9. **controlHub.cs** - script which contains all controls variables

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

Let’s look a little closer

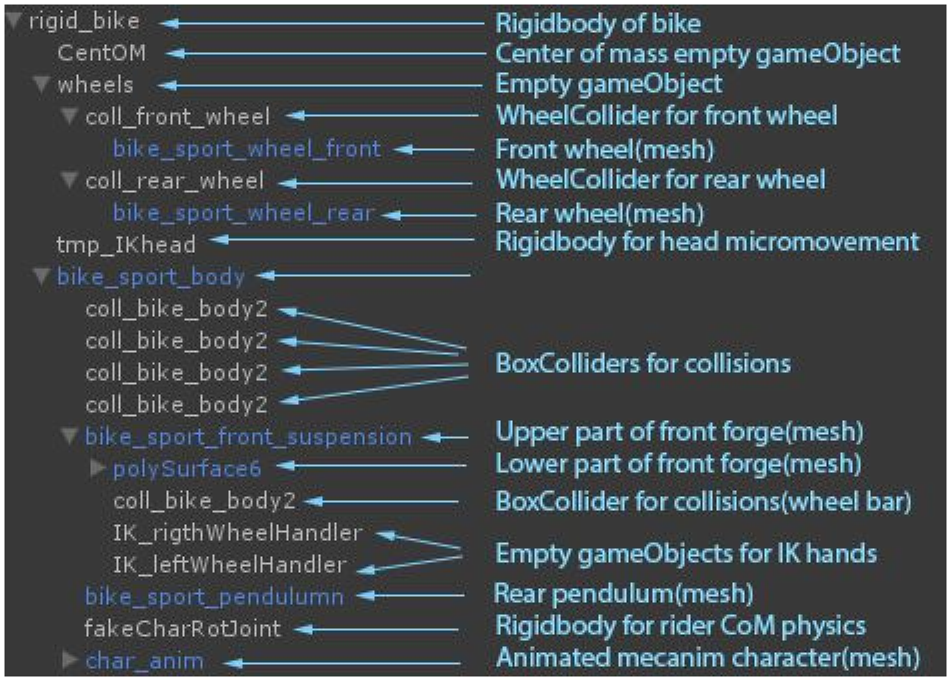
1. **Three bikes and five scenes**(most complex part of manual)

This models is just a **example** how your motorcycle 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 bike from 3d modeling package just **follow** the those **principles of cutting** model: **body, two parts of front forge, pendulum, two wheels.** 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 bike after fall.

The main script **BikeController.cs**

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 bike with that code you take **your models(body, front forge, rear pendulum, two wheels)** and construct the **same tree** as on the picture:

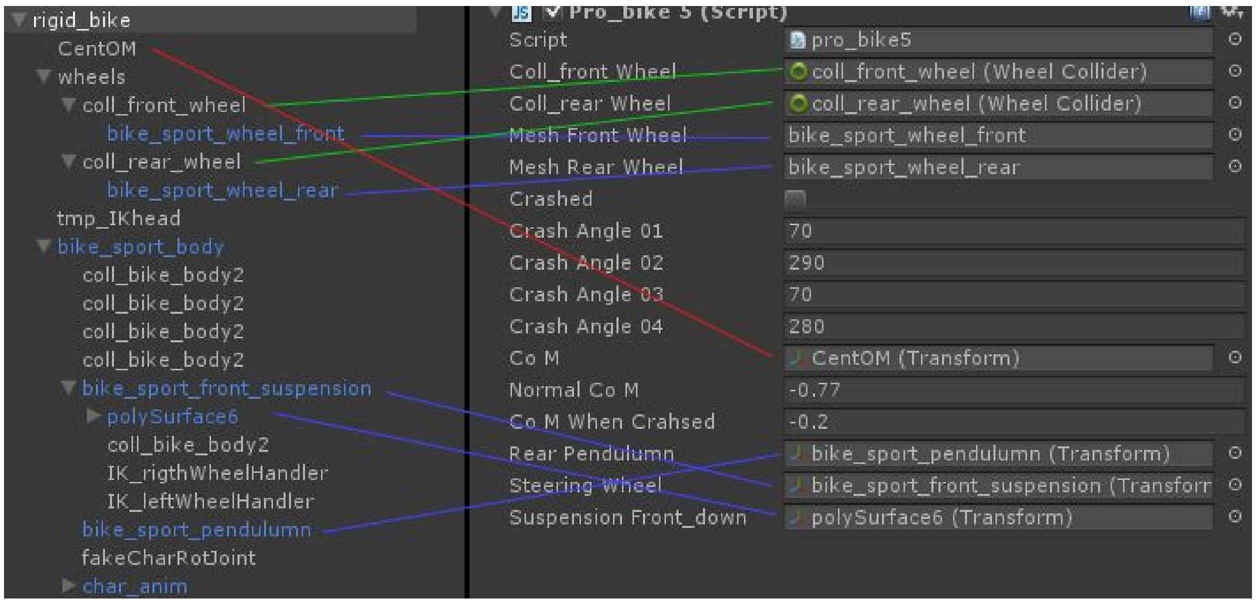


```
▼ rigid_bike
    CentOM
  ▼ wheels
    ▼ coll_front_wheel
        bike_sport_wheel_front
    ▼ coll_rear_wheel
        bike_sport_wheel_rear
    tmp_IKhead
  ▼ bike_sport_body
      coll_bike_body2
      coll_bike_body2
      coll_bike_body2
      coll_bike_body2
    ▼ bike_sport_front_suspension
      ▶ polySurface6
        coll_bike_body2
        IK_rigthWheelHandler
        IK_leftWheelHandler
      bike_sport_pendulumn
      fakeCharRotJoint
    ▶ char_anim
```

Rigidbody of bike
Center of mass empty gameObject
Empty gameObject
WheelCollider for front wheel
Front wheel(mesh)
WheelCollider for rear wheel
Rear wheel(mesh)
Rigidbody for head micromovement
BoxColliders for collisions
Upper part of front forge(mesh)
Lower part of front forge(mesh)
BoxCollider for collisions(wheel bar)
Empty gameObjects for IK hands
Rear pendulum(mesh)
Rigidbody for rider CoM physics
Animated mecanim character(mesh)

rigid_bike settings:
First of all, make sure to **change weight** of rigid body to real weight of motorcycle multiply by 2(new PhysX 3.3 “requirements”). In case of this sportbike its 400kg(200 * 2).
Everything else(drag and angular drag) is doesn’t matter because it’s controlled by script.

Make a connections like it show below:



```
rigid_bike
  CentOM
  wheels
    coll_front_wheel
      bike_sport_wheel_front
    coll_rear_wheel
      bike_sport_wheel_rear
  tmp_IKhead
  bike_sport_body
    coll_bike_body2
    coll_bike_body2
    coll_bike_body2
    coll_bike_body2
    bike_sport_front_suspension
      polySurface6
      coll_bike_body2
      IK_rigthWheelHandler
      IK_leftWheelHandler
    bike_sport_pendulumn
    fakeCharRotJoint
    char_anim
```

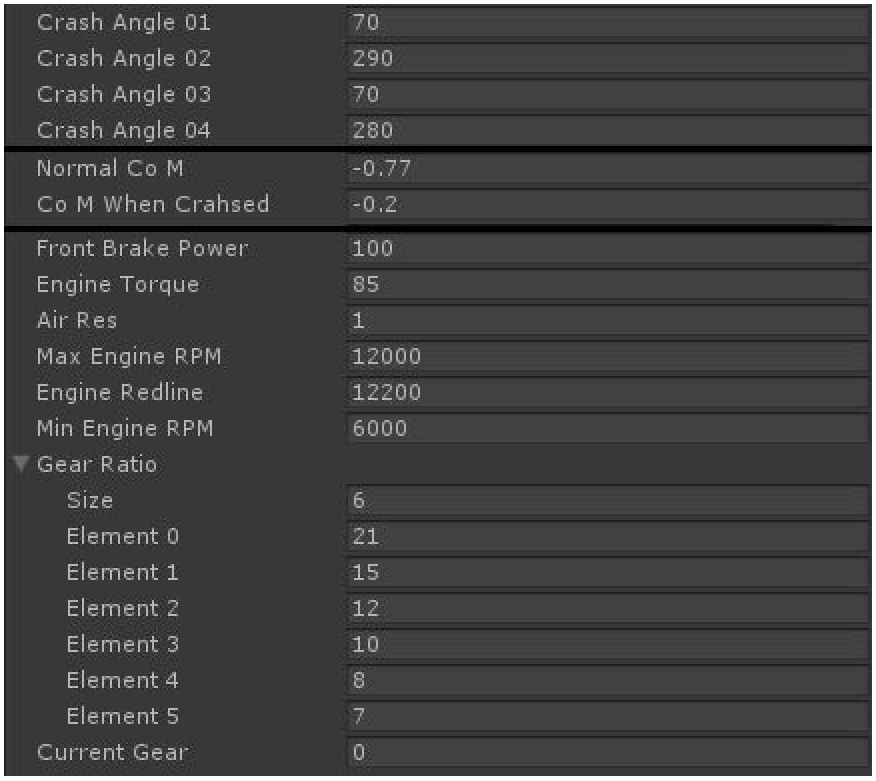
Pro_bike 5 (Script)

Script	pro_bike5
Coll_front Wheel	coll_front_wheel (Wheel Collider)
Coll_rear Wheel	coll_rear_wheel (Wheel Collider)
Mesh Front Wheel	bike_sport_wheel_front
Mesh Rear Wheel	bike_sport_wheel_rear
Crashed	
Crash Angle 01	70
Crash Angle 02	290
Crash Angle 03	70
Crash Angle 04	280
Co M	CentOM (Transform)
Normal Co M	-0.77
Co M When Crahsed	-0.2
Rear Pendulumn	bike_sport_pendulumn (Transform)
Steering Wheel	bike_sport_front_suspension (Transforr
Suspension Front_down	polySurface6 (Transform)

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 bike tree, place Center of Mass empty gameobject just few centimeters **above** the line where wheel touches ground.
Rear Pendulumn - link to bike_sport_pendulum mesh(FBX). This pendulum looks at rear wheel.
Steering Wheel - link to wheel_vis(FBX). So, it’s wheelbar of front forge.
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.

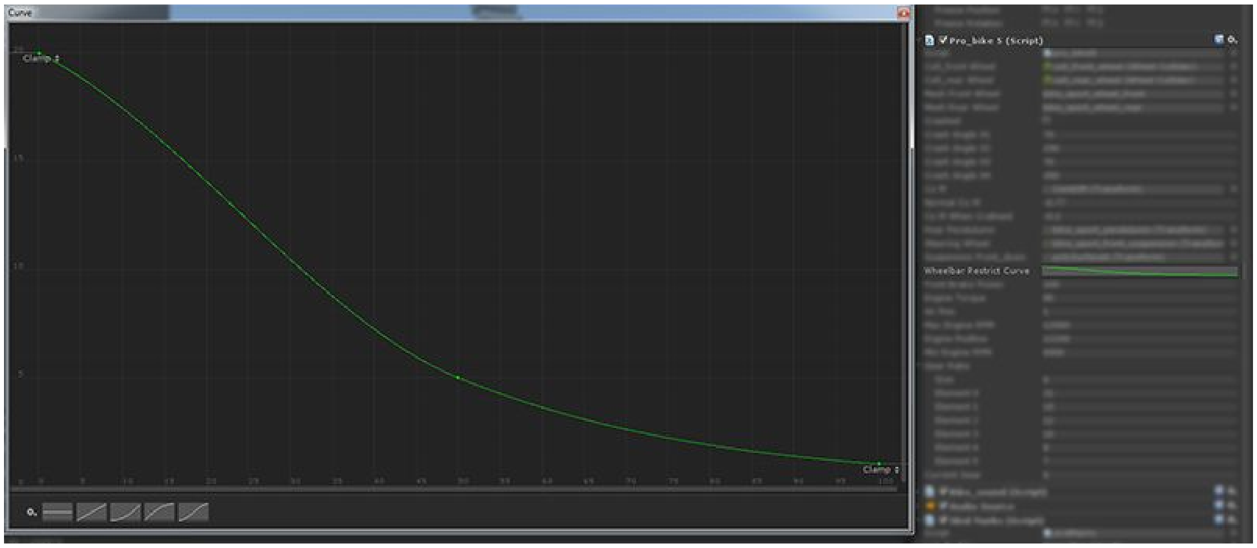
Time to explain some magic numbers :)



Crash Angle 01	70
Crash Angle 02	290
Crash Angle 03	70
Crash Angle 04	280
Normal Co M	-0.77
Co M When Crahsed	-0.2
Front Brake Power	100
Engine Torque	85
Air Res	1
Max Engine RPM	12000
Engine Redline	12200
Min Engine RPM	6000
Gear Ratio	
Size	6
Element 0	21
Element 1	15
Element 2	12
Element 3	10
Element 4	8
Element 5	7
Current Gear	0

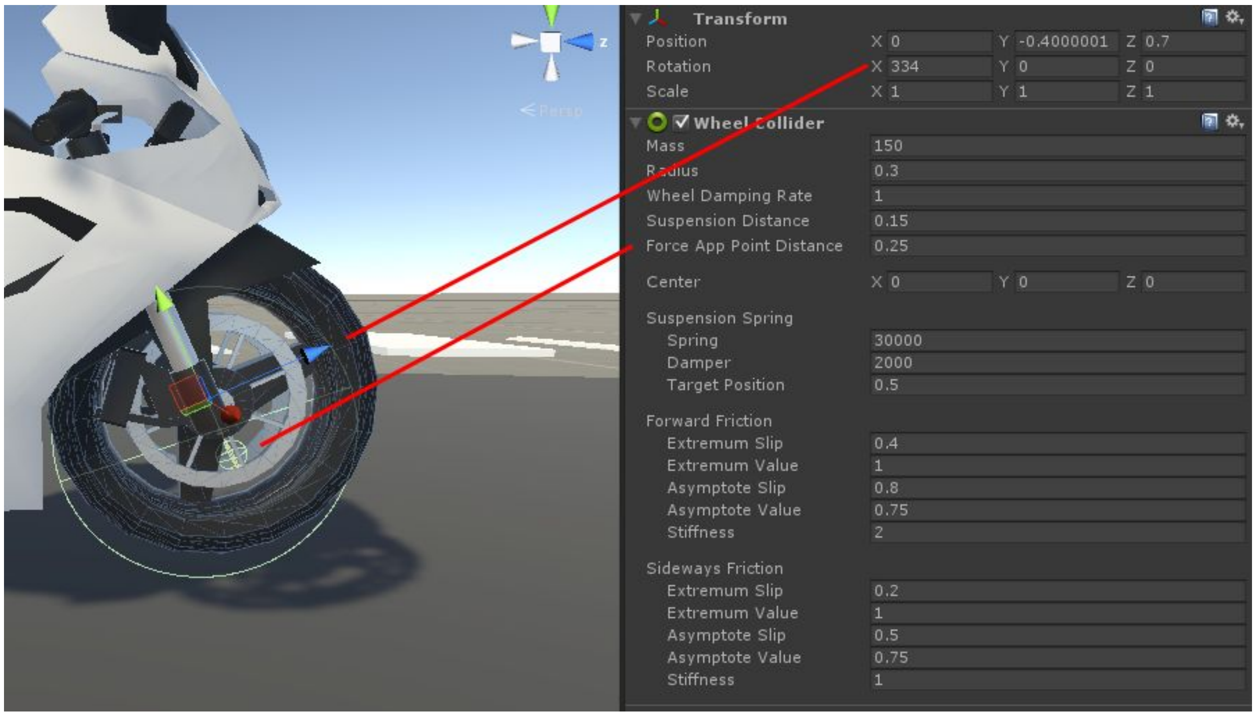
Crash Angle 01-04: it's angles when bike takes "crashed" status - too much lean. 01-02 is for right/left angle. 03-04 is for front(stoppie)/rear(wheelie) angle.
Normal Co M(normal Center of mass): it's CoM Y coordinate. You need experiments to find fine CoM for your own bike.
Co M When Crashed: you need put up CoM when bike crashes. Just for funny turning when crashed.
Front Brake Power: this means very good Brambo brakes for bike.
Engine Torque: it's not exactly horsepowers but very similar.
Air Res: it's for air resistance at high speed. 1 is neutral. Make more for really big bikes and smaller for pitbikes, motocross and so on.
Max Engine RPM: as it sounds.
Engine Redline: as it sounds. Nothing to add.
Min Engine RPM: RPM when gearbox shifted down one gear.
Gear Ratio Size: number of gears in gearbox. For example 5 used for chopper.
Element0-5: you need to find proper numbers for your bike.
Current Gear: it's external variable to use in other scripts.

Now, last trick in that script.
In real life controls of bike at slow speeds and high speeds absolutely different. That calls “countersteering”. This is physics of real life. Here, to achieve this effect and make bike more stable on high speed we need to clamp wheelbar angle according the speed. It means - the faster bike rides the less angle you can rotate wheel bar.
This is example curve for sportbike. Y(vertical) - wheelbar angle, X(horizontal) - bike speed



For your motorcycle tune this curve manually in Editor to get something like that:
at speed 0 km/h = 20 degree, 17 for chopper (25 is VERY turnable bike like motard or something like motocross)
at speed 10 km/h = 18 sport, 15 for chopper
at speed 20 km/h = 14 sport, 12 for chopper
at speed 30 km/h = 9 sport, 8 for chopper
at speed 40 km/h = 6 sport, 5.5 for chopper
at speed 50 km/h = 4 sport, 4 for chopper
at speed 60 km/h = 3 sport, 2 for chopper
at speed 70 km/h = 2.5 sport, 1.4 for chopper
at speed 80 km/h = 2 sport, 0.9 for chopper
at speed 90 km/h = 1.5 sport, 0.6 for chopper
at speed 100 km/h = 1 sport, 0.35 for chopper
at speed over 100 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 100km/h so, last meaning applies to speed over 100km/h

And last thing you should know is - **wheelCollider** settings.
This is good **wheelCollider settings** for front wheel of sportbike(play with settings for chopper or so):



```
Transform
Position                  X 0        Y -0.4000001   Z 0.7
Rotation                  X 334      Y 0            Z 0
Scale                     X 1        Y 1            Z 1
Wheel Collider
Mass                      150
Radius                    0.3
Wheel Damping Rate        1
Suspension Distance       0.15
Force App Point Distance  0.25

Center                    X 0        Y 0            Z 0

Suspension Spring
  Spring                  30000
  Damper                  2000
  Target Position         0.5

Forward Friction
  Extremum Slip           0.4
  Extremum Value          1
  Asymptote Slip          0.8
  Asymptote Value         0.75
  Stiffness               2

Sideways Friction
  Extremum Slip           0.2
  Extremum Value          1
  Asymptote Slip          0.5
  Asymptote Value         0.75
  Stiffness               1
```

Mass 150kg for a wheel is another new PhysX 3 “requirements”.
Pay attention for **Rotation X 334**. It’s 26(360-334 = 26)degree 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 bike.

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

That’s all you need to know about bike settings. Other settings(not necessary, like sounds and skidmarks) will be explain later.

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

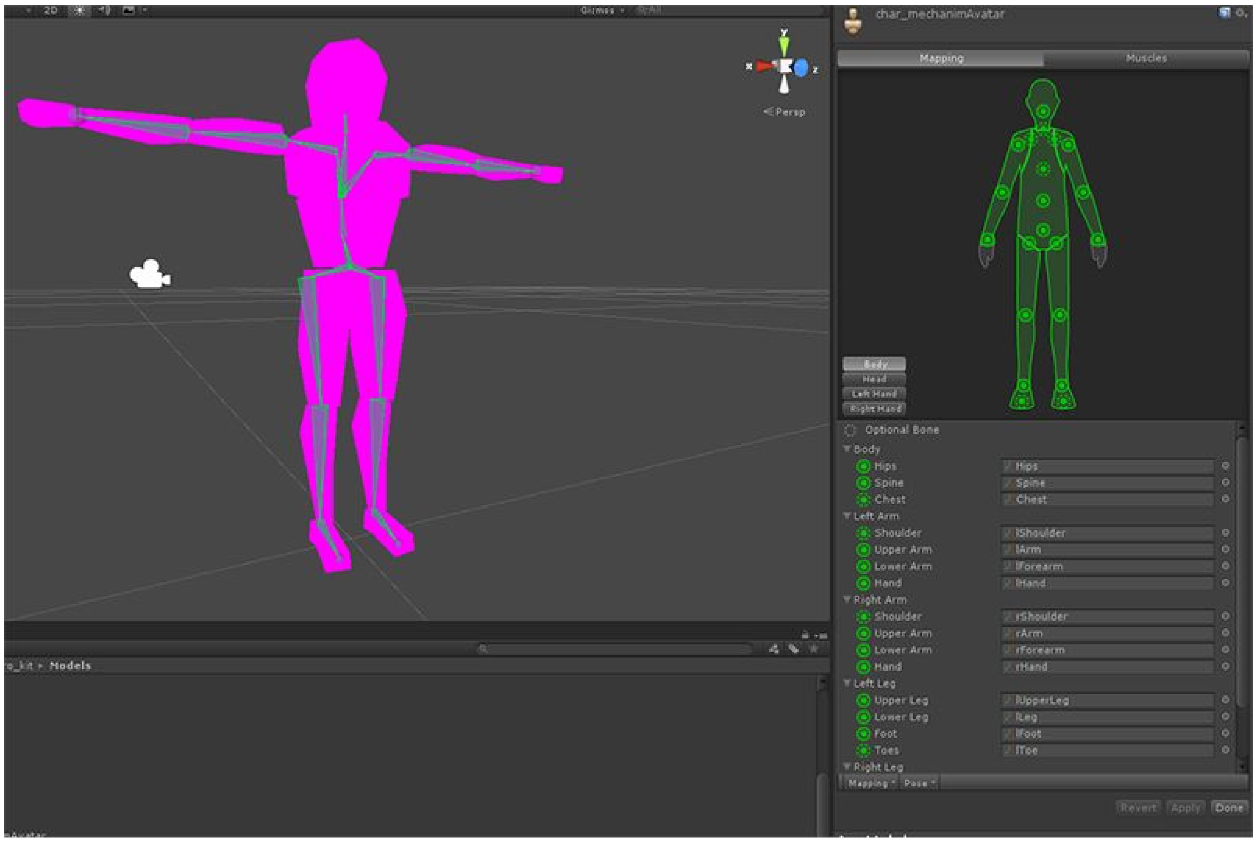
In that section I’ll explain how to set up character as a rider.

You need **3.fbx** files:

a. **char_mechanim.fbx**
b. **char_anim.fbx**
c. **char_anim_chop.fbx**

There is meshes for biker. First - char_mechanim.fbx is just mesh for mecanim setup. Next three is animated meshes(char_anim.fbx, char_anim_chop.fbx and char_anim_motard.fbx)

The char_mechanim.fbx is mapped as usual mecanim avatar from mecanim manual: http://docs.unity3d.com/Manual/ConfiguringtheAvatar.html

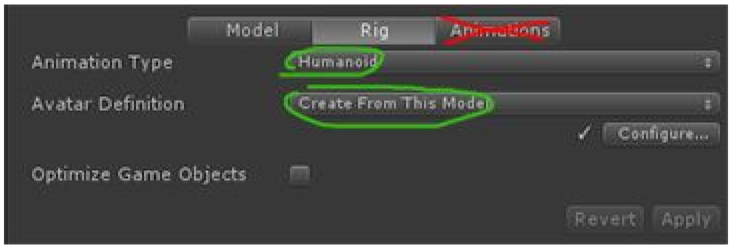


Optional Bone

Body	
Hips	Hips
Spine	Spine
Chest	Chest
Left Arm	
Shoulder	lShoulder
Upper Arm	lArm
Lower Arm	lForearm
Hand	lHand
Right Arm	
Shoulder	rShoulder
Upper Arm	rArm
Lower Arm	rForearm
Hand	rHand
Left Leg	
Upper Leg	lUpperLeg
Lower Leg	lLeg
Foot	lFoot
Toes	lToe
Right Leg	

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”



Animation Type: Humanoid
Avatar Definition: Create From This Model
✓ Configure...
Optimize Game Objects

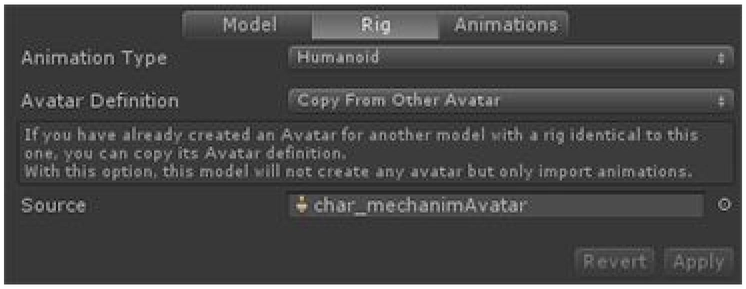
Now, time for **char_anim.fbx**, **char_anim_chop.fbx** and **char_anim_motard.fbx.** The char_anim_chop.fbx(and char_anim_motard.fbx as well) is same as char_anim.fbx but with other animations(pose and leaning is different on sport and chopper/motard bikes). So we won't talk about second "chop" and third "motard" characters. The settings absolutely same as "sport".

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 bike 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.



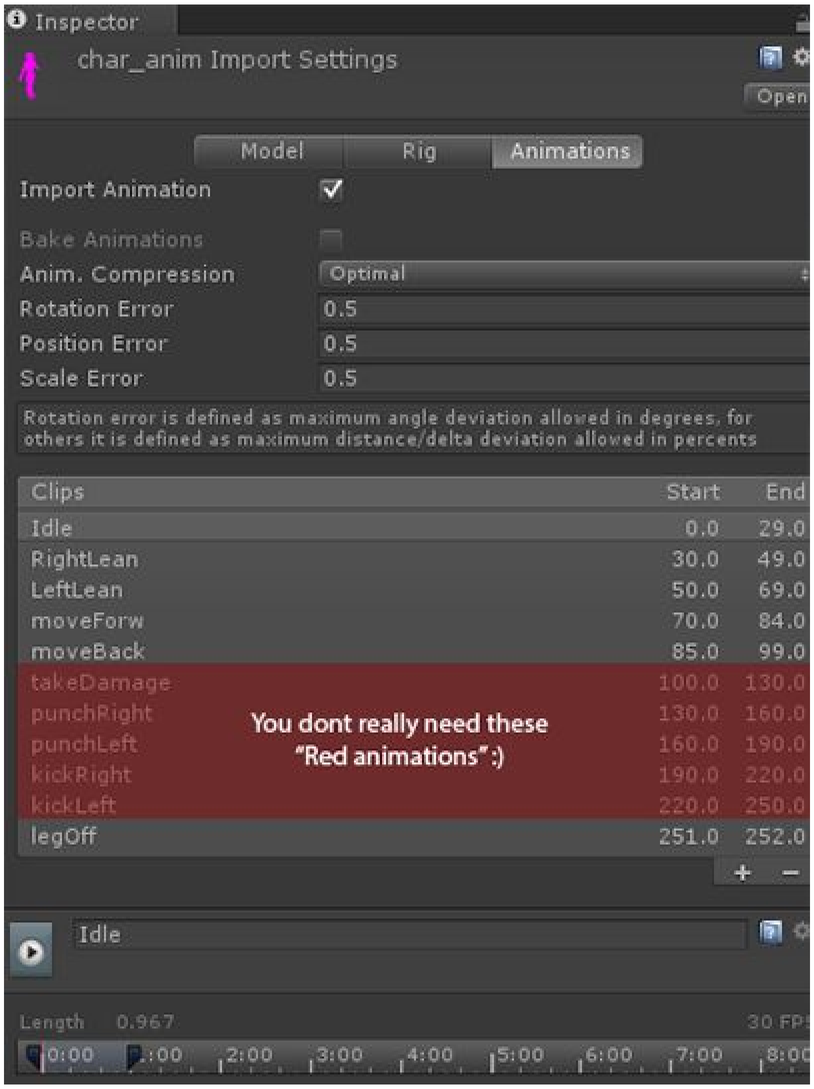
Animation Type: Humanoid
Avatar Definition: Copy From Other Avatar

If you have already created an Avatar for another model with a rig identical to this one, you can copy its Avatar definition.
With this option, this model will not create any avatar but only import animations.

Source: char_mechanimAvatar

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)



Import Animation ✓
Bake Animations
Anim. Compression: Optimal
Rotation Error: 0.5
Position Error: 0.5
Scale Error: 0.5

Rotation error is defined as maximum angle deviation allowed in degrees, for others it is defined as maximum distance/delta deviation allowed in percents

Clips	Start	End
Idle	0.0	29.0
RightLean	30.0	49.0
LeftLean	50.0	69.0
moveForw	70.0	84.0
moveBack	85.0	99.0
takeDamage	100.0	130.0
punchRight	130.0	160.0
punchLeft	160.0	190.0
kickRight	190.0	220.0
kickLeft	220.0	250.0
legOff	251.0	252.0

You dont really need these “Red animations” :)

Idle

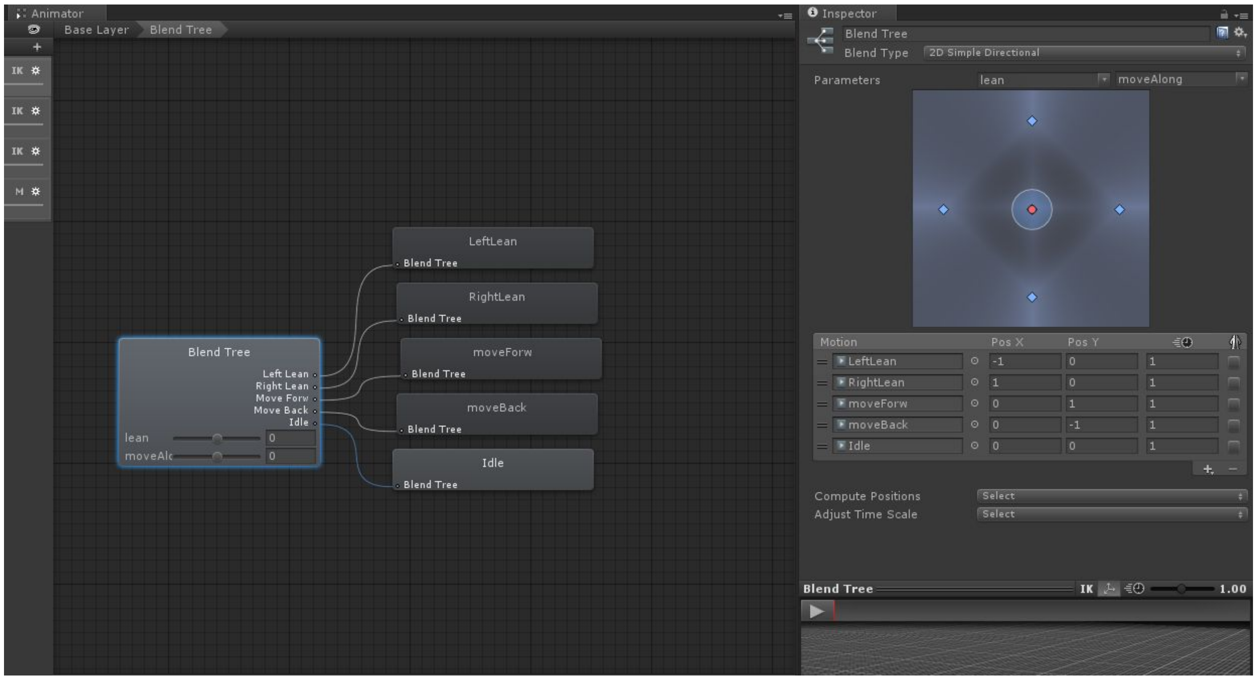
Length 0.967 30 FP

there is some more fight animation :) don’t pay attention for this. Just for fun

Do NOT forget to check **“Loop Time”** for **idle** animation :)

Your character is ready to play animations and ride a bike. 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 **BikerAnim.cs**

1. Drag your character.fbx to rigid_bike/body(as you read before in chapter “Two bikes and three scenes”).
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



Blend Tree
Blend Type: 2D Simple Directional
Parameters: lean, moveAlong

Motion	Pos X	Pos Y		
LeftLean	-1	0	1	
RightLean	1	0	1	
moveForw	0	1	1	
moveBack	0	-1	1	
Idle	0	0	1	

Compute Positions: Select
Adjust Time Scale: Select

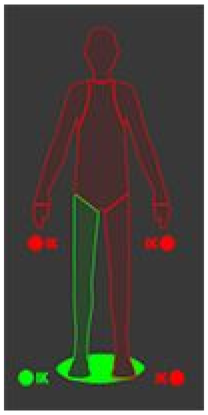
Blend Tree IK 1.00

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 “**BikerAnim.cs**” 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:

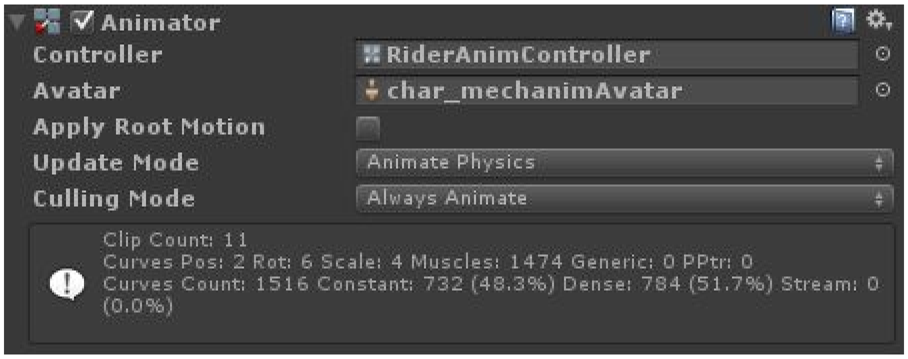


Just one leg is green(every other bones will play their own animations but leg will play “to the ground”).

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.

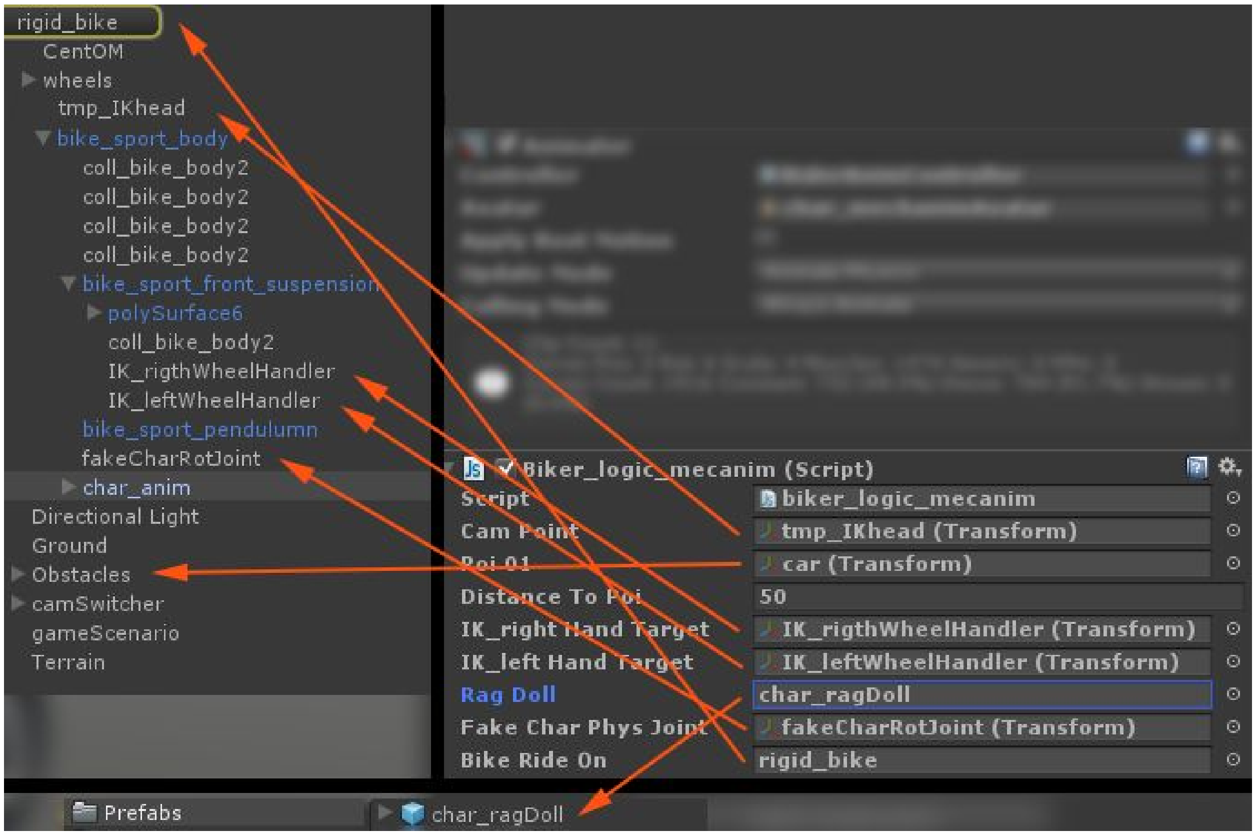


Animator
Controller: RiderAnimController
Avatar: char_mechanimAvatar
Apply Root Motion: ☐
Update Mode: Animate Physics
Culling Mode: Always Animate

```
Clip Count: 11
Curves Pos: 2 Rot: 6 Scale: 4 Muscles: 1474 Generic: 0 PPtr: 0
Curves Count: 1516 Constant: 732 (48.3%) Dense: 784 (51.7%) Stream: 0
(0.0%)
```

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 **BikerAnim.cs** to new character and link some variables as it shown on picture below:



```
rigid_bike
  CentOM
  ▶ wheels
    tmp_IKhead
  ▼ bike_sport_body
      coll_bike_body2
      coll_bike_body2
      coll_bike_body2
      coll_bike_body2
    ▼ bike_sport_front_suspension
      ▶ polySurface6
        coll_bike_body2
        IK_rigthWheelHandler
        IK_leftWheelHandler
    bike_sport_pendulumn
    fakeCharRotJoint
  ▶ char_anim
Directional Light
Ground
▶ Obstacles
▶ camSwitcher
gameScenario
Terrain
```

Biker_logic_mecanim (Script)

Script	biker_logic_mecanim
Cam Point	tmp_IKhead (Transform)
Poi 01	car (Transform)
Distance To Poi	50
IK_right Hand Target	IK_rigthWheelHandler (Transform)
IK_left Hand Target	IK_leftWheelHandler (Transform)
Rag Doll	char_ragDoll
Fake Char Phys Joint	fakeCharRotJoint (Transform)
Bike Ride On	rigid_bike

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. Few sounds for bikes

Those sounds is:

sport_engine_start, sport_engine_idle, sport_engine_high, skid, gears_change and similar sounds for chopper bike.

Pay attention for “idle” and “high” - there is trick will be explained in **BikeAudio.cs** section.

Just want to say again - my sounds is royalty free and quite bad quality. Some of them recorded car engine not even bike. Find better ones and your bike will buuuuurn ears !

4. BikeController.cs

There is main script for each bike in project. See detailed info in comments in script. By that script you may control any bike 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. BikeAudio.cs

You should add AudioSource to your bike.

Put this script on rigid_bike.

Then link sounds you want to these variables:

Engine Start Sound for ignition sound. For every new start bike from restart(crash)

Gearing Sound for sound of changing gear

Idle RPM for engine sound when idle and(!) and low RPM(engine rotates per minute)

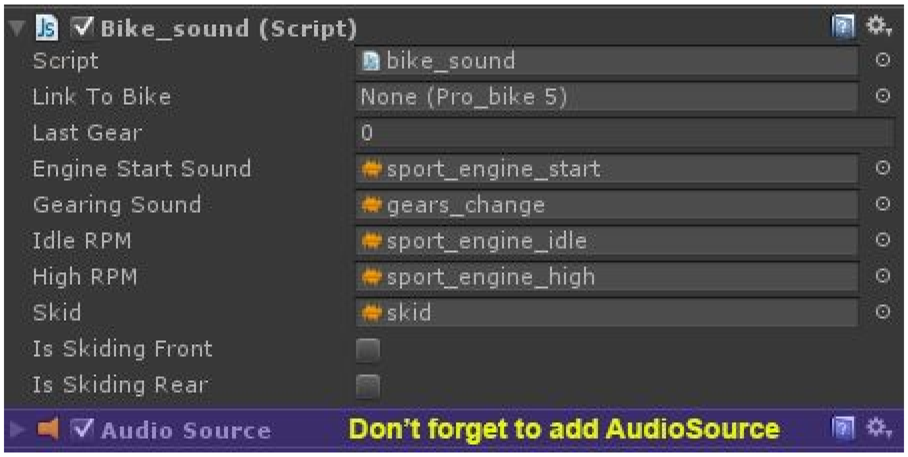
High RPM for high RPM

Why you need two sounds for low RPM and high RPM ?

That’s why engine sound isn’t just flat sound which becomes higher when more accelerate. The sound is complex and very different at 1.000RPM and 10.000RPM. That’s why there is two sounds which mixed at 50% of RMP. The Idle RPM sound plays for 100% volume at low RPM and high RPM sound plays for 0% volume same time. But when you press accelerate button, volume of Idle RPM put down to 0% and high RPM volume grows up same time to 100%.

So, there is easy way to add to script “inbetween” sounds to recreate any quality of real bike sounds. All you need is high quality sounds. So, two mixed sounds is just example. Make 5 of them, 10.. for any recorded RPM you have.

Last sound is **skid** for skidding sound. Nothing to explain.


Bike_sound (Script)

Script	bike_sound
Link To Bike	None (Pro_bike 5)
Last Gear	0
Engine Start Sound	sport_engine_start
Gearing Sound	gears_change
Idle RPM	sport_engine_idle
High RPM	sport_engine_high
Skid	skid
Is Skiding Front	
Is Skiding Rear	

Audio Source

Don't forget to add AudioSource

6. BikerAnim.cs

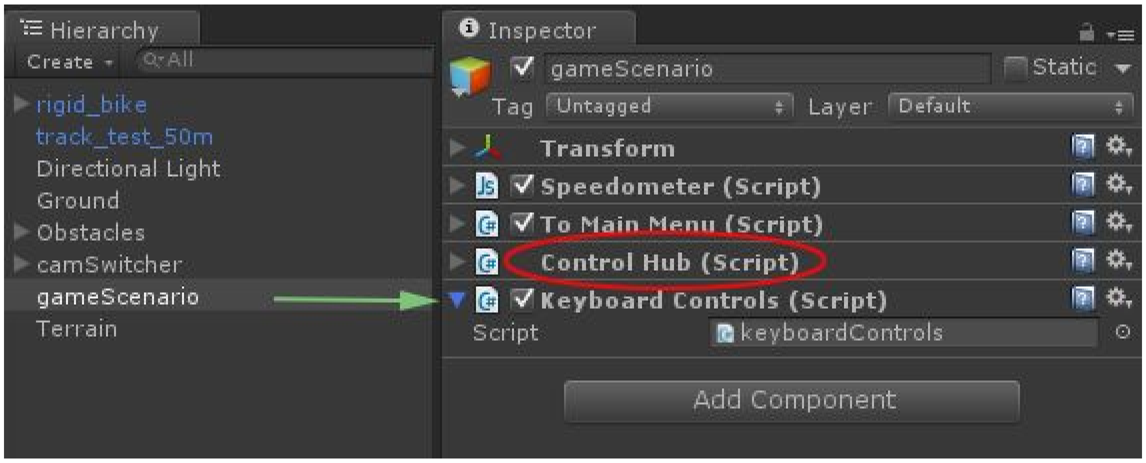
This controls the rider behavior.

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

7. 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 bike controls. This complex scheme made for comfortable change of controller type.

Put this script to empty gameObject called "gameScenario":

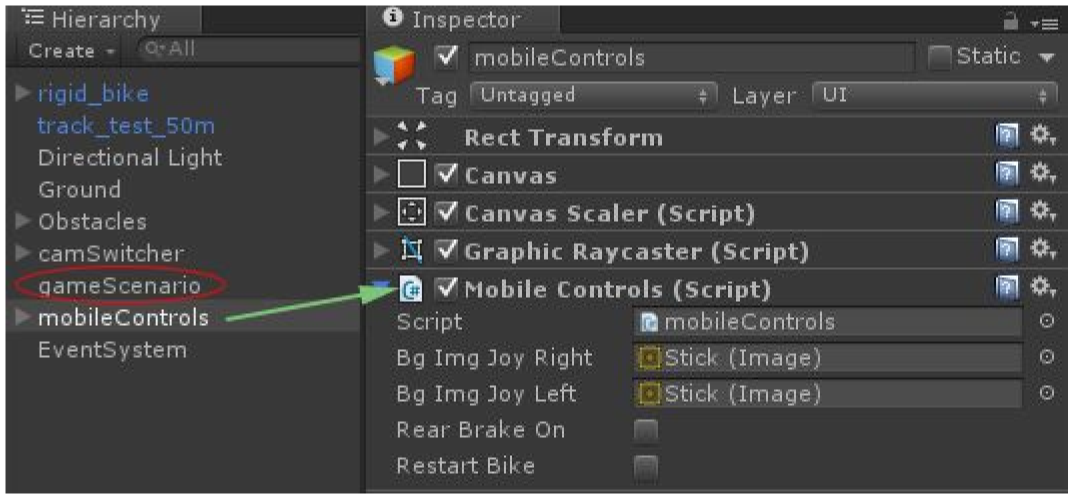


Make sure the "gameScenario" contains script "**controlHub.cs**" as well.

8. 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 bike controls. This complex scheme made for comfortable change of controller type.

Put this script in mobileControls(GameObject/UI/Canvas) canvas type UI element:



mobileControls Static
Tag Untagged Layer UI

Script	mobileControls
Bg Img Joy Right	Stick (Image)
Bg Img Joy Left	Stick (Image)
Rear Brake On	☐
Restart Bike	☐

Make sure the “gameScenario” contains script “**controlHub.cs**”.

9. controlHub.cs

This script only contains empty variables which ones can be read by other scripts(**BikeController.cs** for example). These variables may be changed from any other scripts. The sequence looks like: controlsScripts->**controlHub.cs-**>execution scripts(bike, 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 bike controled by any device: joystick, leap motion and so on.

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

10. 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).

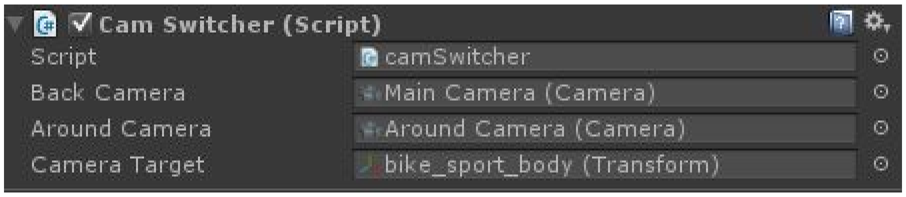
There is empty gameObject named “**camSwitcher**” which include both cameras.

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)



Cam Switcher (Script)

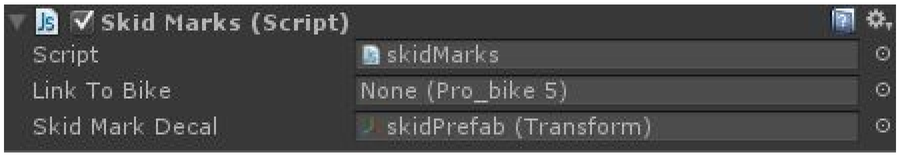
Script	camSwitcher
Back Camera	Main Camera (Camera)
Around Camera	Around Camera (Camera)
Camera Target	bike_sport_body (Transform)

Now camera leaning with biker. It’s more dynamic and realistic. But if you don’t want this effect just delete this code(it’s commented with “------------------” lines).

11. BikeSkid.cs

This script generates skidmarks when bike 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 bike script.



Skid Marks (Script)

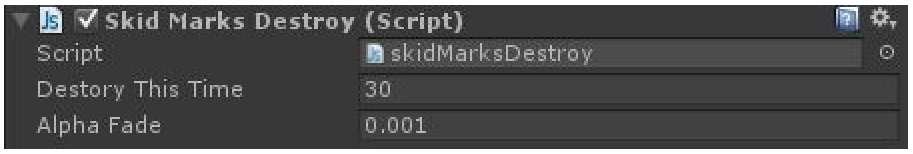
Script	skidMarks
Link To Bike	None (Pro_bike 5)
Skid Mark Decal	skidPrefab (Transform)

12. SkidMarksDestroy.cs

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.



Skid Marks Destroy (Script)

Script	skidMarksDestroy
Destory This Time	30
Alpha Fade	0.001

13. SpeedoMeter.cs

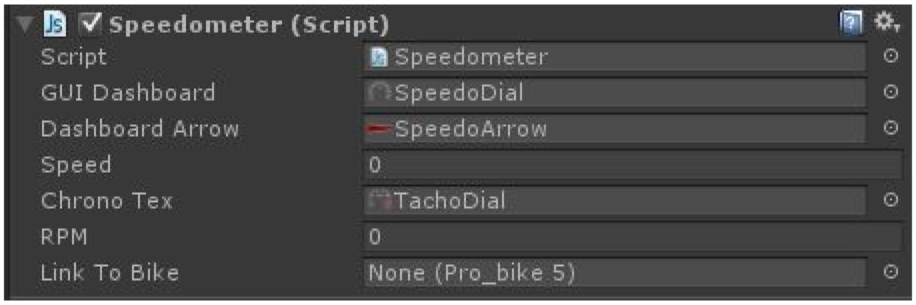
This script is on empty gameObject called “gameScenario”. You need this to print nice GUI texture over screen and see animations of arrows for speedometer and tachometer.

GUI Dashboard load here 2d texture of your speedometer

Dashboard Arrow load here 2d texture of arrow for speedo and tacho

Chrono Tex load here 2d texture of your tachometer

Link to Bike drag here **BikeController.cs** - the speedometer script takes RPM, current gear and bike’s speed from original **BikeController.cs** script.



Speedometer (Script)

Script	Speedometer
GUI Dashboard	SpeedoDial
Dashboard Arrow	SpeedoArrow
Speed	0
Chrono Tex	TachoDial
RPM	0
Link To Bike	None (Pro_bike 5)

14. startScript.cs

This script is for scene **“u5bike_MainMenu”**. It’s used only for selection of bike you want to ride. Nothing interesting to explain.

15. toMainMenu.cs

This script is on empty gameObject called “gameScenario”. Just pings “Escape” key to make back to **“u5bike_MainMenu”** for bike selection. Delete it for sure if you doesn’t need it. Nothing interesting to explain.

Version changes:

1.0 - first release
1.1 - front suspension reworked + minor fixes
1.2 - added motard bike + fixes
1.3 - added reverse speed + fixes
1.4 - rework inertia tensor to make it compatible with Unity 5.3
1.5 - added mobile controls
1.6 - converted to C#(1.6 fixes)

That’s all.
Feel free to ask me anything about that kit by e-mail: smokerr@mail.ru
Boris Chuprin 2018