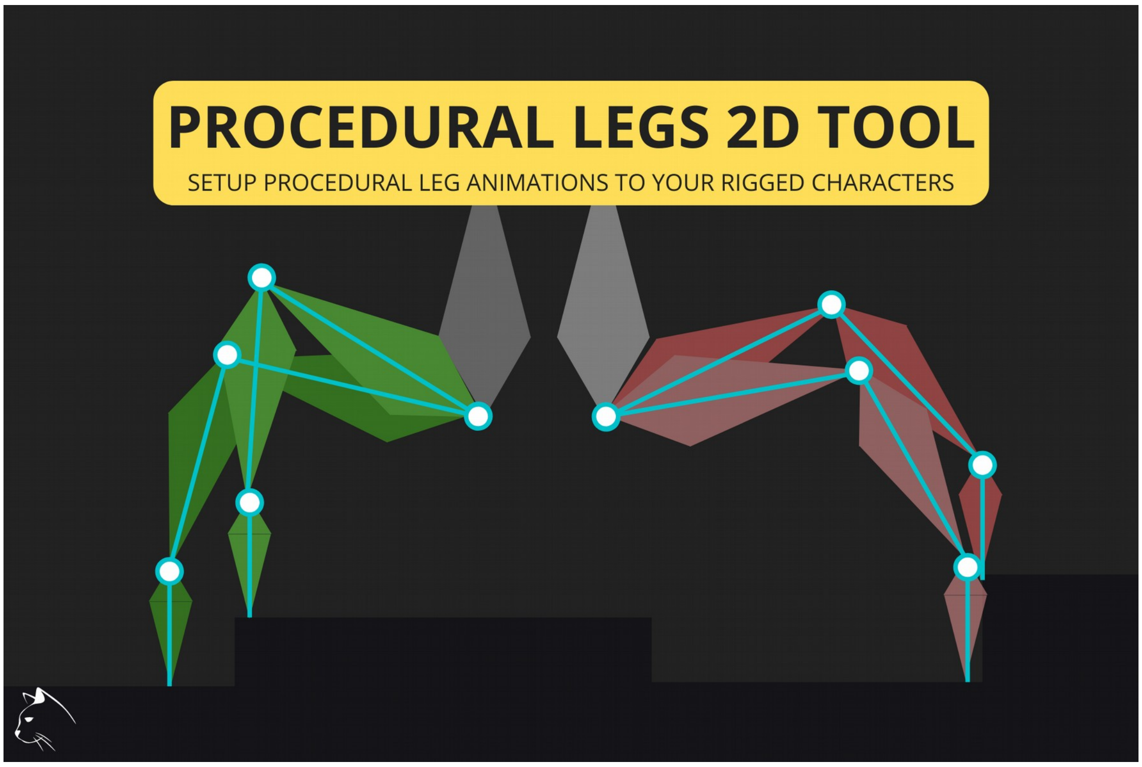
Procedural Legs 2D Tool



Add procedural movement to a 2D rigged lower body (Legs) by simply setting some variables and curves using the **Procedural Legs 2D Tool** interface. Make multiple walk like movements (ex.: walk, run, crouch, fast walk) that fit the same body to be used and selected during the gameplay.

http://u3d.as/1Z8C

This documentation contains the detailed explanation of the implemented features, classes, methods, variables, concepts and setting adjustments with an step by step set up guide of, setting up the body and animations.

Contact

assetstore.unity.com/publishers/19535

github.com/danielcmcg

d.cavalcante.m@gmail.com

Daniel C Menezes
http://danielcmcg.github.io

1 CONTENTS

2 OVERVIEW

The **Procedural Legs 2D Tool (PL2D)** helps the creation of different walking or run styles on 2D rigged body legs (with bones or hierarchically only). The legs movement are procedural and uses a combination of IK and positioning calculations.

The system provide a way to save many different settings for the leg’s movement into a PL2D Animation (examples: “normal walk”, “stealth walk”, “run”), these can be viewed and edited using the PL2D Animator component.

The sections below explain the main features of the system and the user interface, there is also, the last section, a step by step guide on how to set up the body and get the animation working on a body.

If you are comfortable, you can jump right to the step by step guide for a quick tutorial. But remember to come back to the previous sections if you encounter some difficulties.

This project is in continuous development. If you have any suggestion for adding, improving, simplifying features or the Animator interface or want to share your view about the asset, feel free to send me an email. Your feedback is valuable for guiding the improvements.

3 BODY RIGGING

For the PL2D system to correctly apply the needed components on the character, its lower body must be composed of Axial Bone(s) and Legs set up hierarchically and indexed by their Axial Bones and Feet names.

3.1 Axial Bones

Axial Bones are the game objects where the legs are attached to. They must be parent of the Legs. Note that a biped is normally composed of one Axial Bone and two Legs (thigh, shin and foot), while a quadruped character is composed of two Axial Bones, each with two child Legs.

An Axial Bone is a concept related to this asset to better define the bone that hold the legs and, if the body has an upper body, is between the upper body and legs. For example, a six legged body can either be sideways to the camera view or facing it, for the first case it would be composed of one Axial Bone on the front with three Legs and another at the end with the other Legs, for the second case, it would have the Axial Bones placed on each side of the character, in both cases the rigging is the same.

That setup of attaching child Legs on a proper parent Axial Bones is needed since the Legs attached to a particular Axial Bone are called to move in sequence.

For further understanding, see the video tutorial on Rigging for Procedural Legs 2D Asset [video tutorial link].

3.2 Legs

The body Legs are composed of leg bones and a foot (the last being the IK end effector). The rigging of the legs is the default IK rigging, where each bone closer to the end effector is a child of the previous, being the end effector the end child.

3.3 Bones Hierarchy and Naming

The Procedural Legs 2D automatically applies the needed components to a character by finding its Axial Bones and Foots by their names when this body is Initialized using the PL2D Animator.

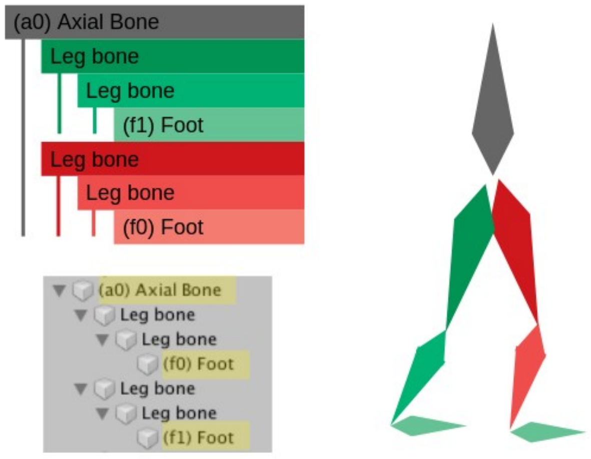
The name of each Axial Bone must start with “(a” plus an int number that is the index of the bone, this index begins with 0 for the first Axial Bone, so it would be named “(a0)”.

The name of each Foot must start with “(f” plus an int number that is the index of the leg, beginning with 0 and added by 1 for each next leg on a particular Axial Bone.

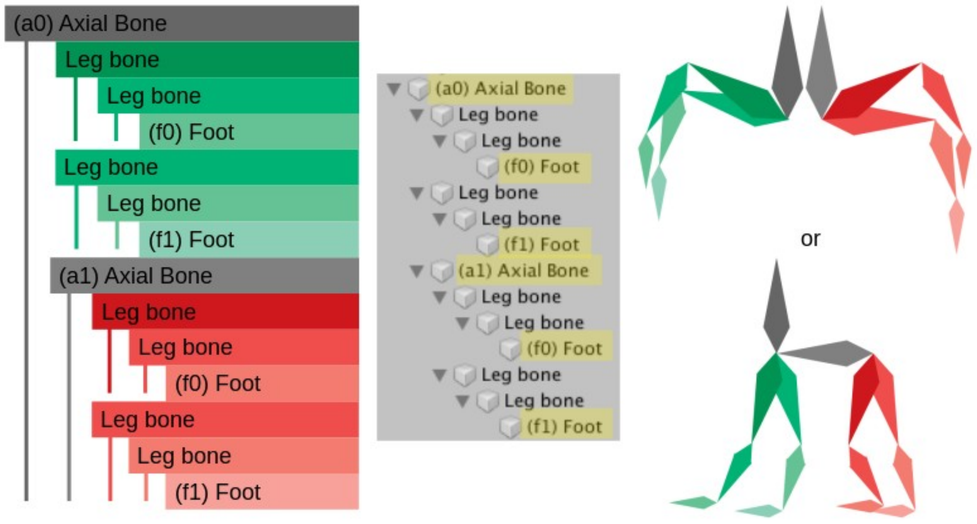
OBS.: The naming rule applies only for the beginning of the name, so it is possible to compose the name, as “(a0) – robot bone”, or “(f0) bone”.

Following are some examples of rigging and naming characters.

Biped



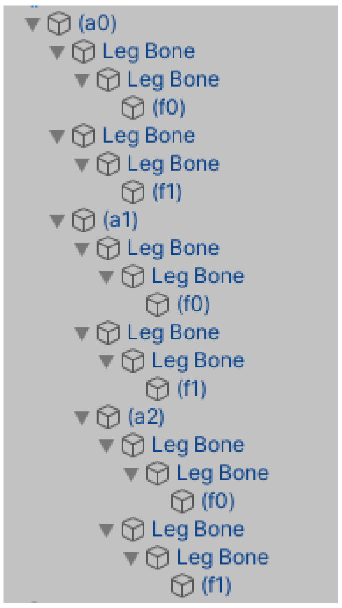
Quadruped



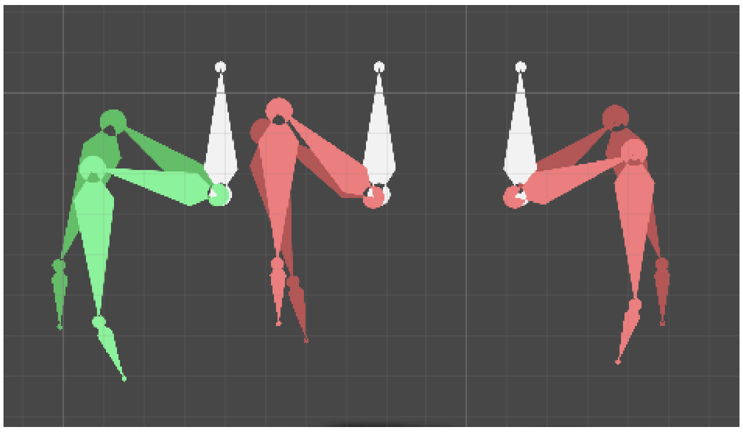
OBS. 1: The Axial Bone may have as many child Legs as needed, also, the Legs can have more than two intermediate bones (for that, use the chain IK solver).

OBS. 2: For consistency of legs sequencing, divide the body into sections (front and back or left and right sides) and make one Axial Body per section.

Three Axial Bones



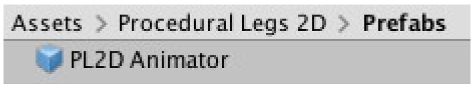
```
▼ (a0)
  ▼ Leg Bone
    ▼ Leg Bone
        (f0)
  ▼ Leg Bone
    ▼ Leg Bone
        (f1)
  ▼ (a1)
    ▼ Leg Bone
      ▼ Leg Bone
          (f0)
    ▼ Leg Bone
      ▼ Leg Bone
          (f1)
    ▼ (a2)
      ▼ Leg Bone
        ▼ Leg Bone
            (f0)
      ▼ Leg Bone
        ▼ Leg Bone
            (f1)
```



OBS. 1: <u>It is not recommended to have a body with more than three Axial Bones if the ground has rough angled surfaces where the body can turn on its own</u>. The Angle solvers can affect each other leading to unstable body.

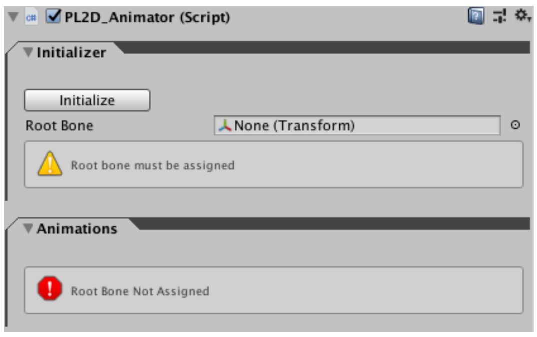
4 PL2D ANIMATOR

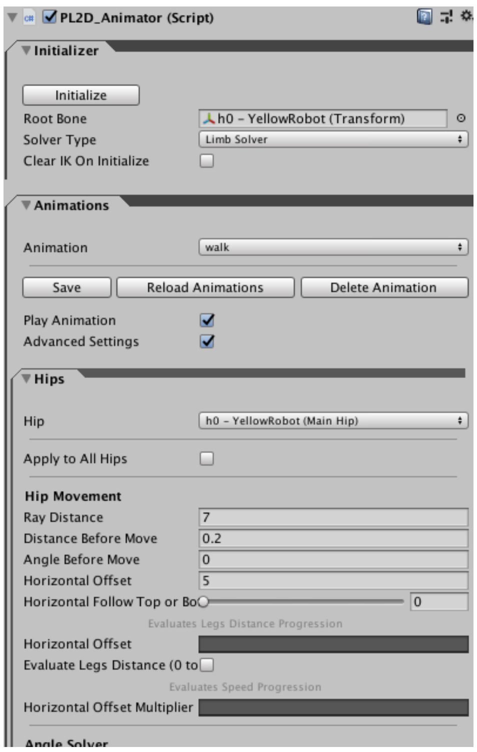
The PL2D Animator contains the main interface of the system. It is a component that can be attached to any GameObject with an unique name (to better organize your project, use the PL2D Animator prefab that is inside the Prefabs folder).


Assets > Procedural Legs 2D > **Prefabs**
PL2D Animator

OBS.: Once you drag the PL2D Animator prefab to the project, it will be automatically unpacked so it can be used independently from other Animators in scene.

The component holds the main interface for setting up the animations (meaning the group of settings needed for the legs to move accordingly) on the body. From there it is possible to initialize a previously rigged body with IK solvers and the needed PL2D components. Once initialized, the interface shows up the settings for, edit, save and load animations.


PL2D_Animator (Script)
Initializer
Initialize
Root Bone: None (Transform)
Root bone must be assigned
Animations
Root Bone Not Assigned


PL2D_Animator (Script)
Initializer
Initialize
Root Bone: h0 - YellowRobot (Transform)
Solver Type: Limb Solver
Clear IK On Initialize: ☐
Animations
Animation: walk
Save | Reload Animations | Delete Animation
Play Animation: ✓
Advanced Settings: ✓
Hips
Hip: h0 - YellowRobot (Main Hip)
Apply to All Hips: ☐
Hip Movement
Ray Distance: 7
Distance Before Move: 0.2
Angle Before Move: 0
Horizontal Offset: 5
Horizontal Follow Top or Bo: 0
Evaluates Legs Distance Progression
Horizontal Offset
Evaluate Legs Distance (0 to ☐
Evaluates Speed Progression
Horizontal Offset Multiplier
Angle Solver

Editing, Saving or loading a particular animation to a body can be done from the PL2D Animator Inspector interface, by selecting an existing animation using the PL2D Animator inspector, or using the PL2D Animator API.

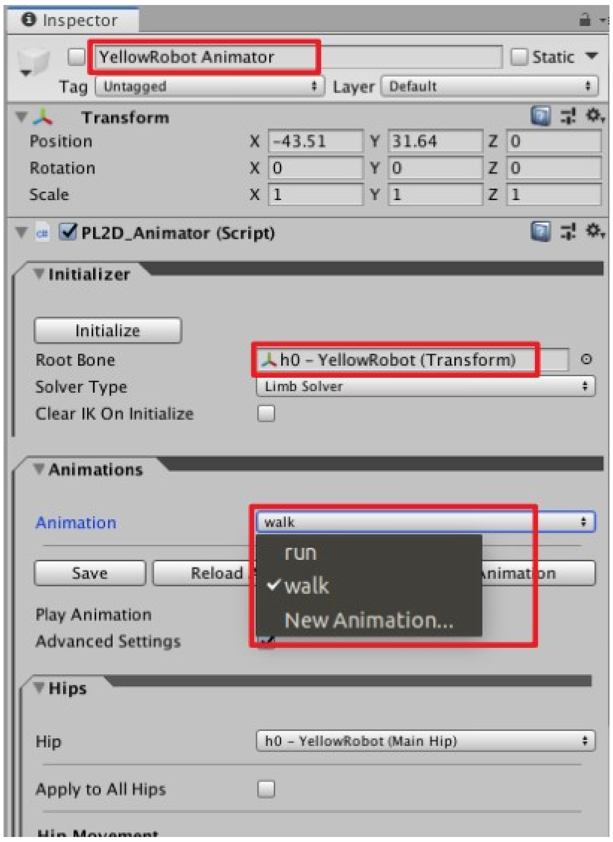
4.1 PL2D Animation

A PL2D Animation is composed of JSON files grouped into a folder of its name. These files contains the Axial Bones and Legs settings and are used by the system to save and load the animations on a body.

Even though one single animator or even a single animation can be used for many bodies (characters on scene), it is important to know that the animation is based on the body structure (body rigging) and store the settings for each indicated Axial Bones and Legs of the body.

Exemplifying, the saved animations of a body with only one Axial Bone and two Legs are going to store the information about those body parts only, so this animations will not be properly loaded on a body with two Axial Bones and six Legs.

Each animation is related to a single PL2D Animator and, for better consistency, is recommended to use one animator (with as many animations as needed) per body.



Inspector

YellowRobot Animator — Static

Tag: Untagged — Layer: Default

Transform

Position	X	-43.51	Y	31.64	Z	0
Rotation	X	0	Y	0	Z	0
Scale	X	1	Y	1	Z	1

PL2D_Animator (Script)

Initializer

Initialize

Root Bone: h0 - YellowRobot (Transform)

Solver Type: Limb Solver

Clear IK On Initialize: ☐

Animations

Animation: walk

Play Animation

Advanced Settings

Hips

Hip: h0 - YellowRobot (Main Hip)

Apply to All Hips: ☐

Hip Movement

The Animations are stored inside the PL2D Animations folder and grouped into its Animator folder. The folders structure is explained in the next section.

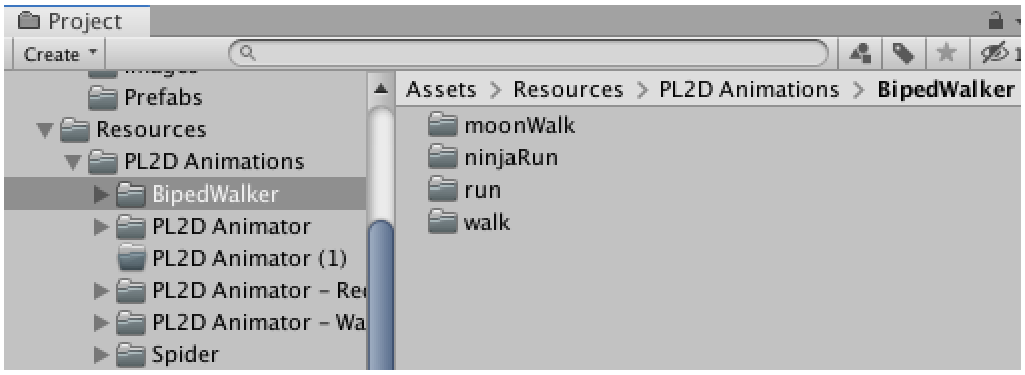
4.2 Resources: Animations Folder

All the animations (animation folder containing the JSON files) are stored inside the PL2D Animations folder, located at "Procedural Legs 2D > Resources > PL2D Animations".

Since the Animations are related to a single Animator, the folder hierarchical structure is the one below:

- PL2D Animations (folder)
    - Animator (folder with the animator name)
        - Animation (folder with the animation name)
            - JSON (files)

As an example, the image below shows the animations inside the BipedWalker Animator folder.

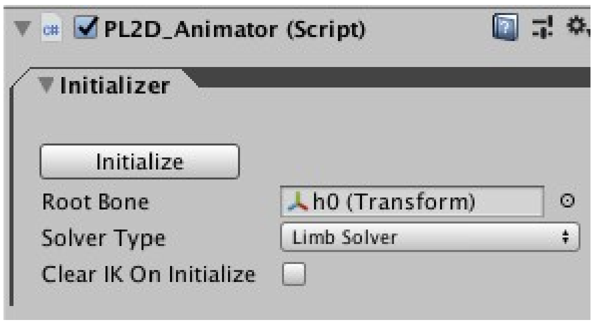


4.3 Inspector Interface

In this section, the PL2D Animator inspector interface is detailed. The Initializer and Animation settings and variables and the interface usage are explained.

4.3.1 Initializer

The Initializer panel holds the interface for the first step on the body setting up. There you indicate which body to initialize and choose the IK options.



PL2D_Animator (Script)

Initializer

Initialize

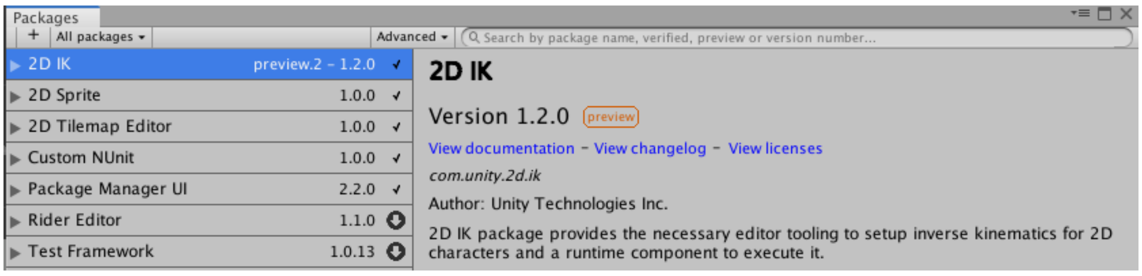
Root Bone	h0 (Transform)
Solver Type	Limb Solver
Clear IK On Initialize	☐

Before initializing the body, by clicking the “Initialize” button, remember to set up the variables explained below.

Root Bone: This variable indicates which body to be initialized, you can select (or drop) the first parent Axial Bone or any parent of it.

Solver Type: In the solver type dropdown you can select between, “Limb Solver”, “Chain Solver” or “Manually Added Solver”. The IK solver uses the indicated Foot as the end effector.

The first two options need the 2D IK Package to be imported on the project (See more on https://docs.unity3d.com/Packages/com.unity.2d.ik@1.1/manual/index.html).



2D IK	preview.2 - 1.2.0 ✓
2D Sprite	1.0.0 ✓
2D Tilemap Editor	1.0.0 ✓
Custom NUnit	1.0.0 ✓
Package Manager UI	2.2.0 ✓
Rider Editor	1.1.0
Test Framework	1.0.13

2D IK

Version 1.2.0 preview

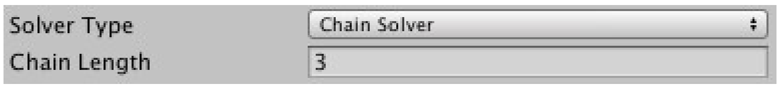
View documentation - View changelog - View licenses

com.unity.2d.ik

Author: Unity Technologies Inc.

2D IK package provides the necessary editor tooling to setup inverse kinematics for 2D characters and a runtime component to execute it.

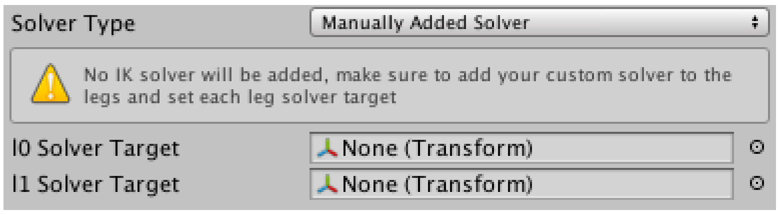
Choosing the “Chain Solver” option, an additional setting (Chain Length) will be shown.



Solver Type	Chain Solver
Chain Length	3

Chain Length: Indicates the IK chain length, counting from the Foot to its parent bones.

The “Manually Added Solver” option requires you to previously add your custom IK solver to the legs and their solver target. By selecting this option, a sequence of placeholders will be displayed for you to set the solver targets, the number of legs will be the number of placeholders that must be set.


Solver Type: Manually Added Solver

No IK solver will be added, make sure to add your custom solver to the legs and set each leg solver target

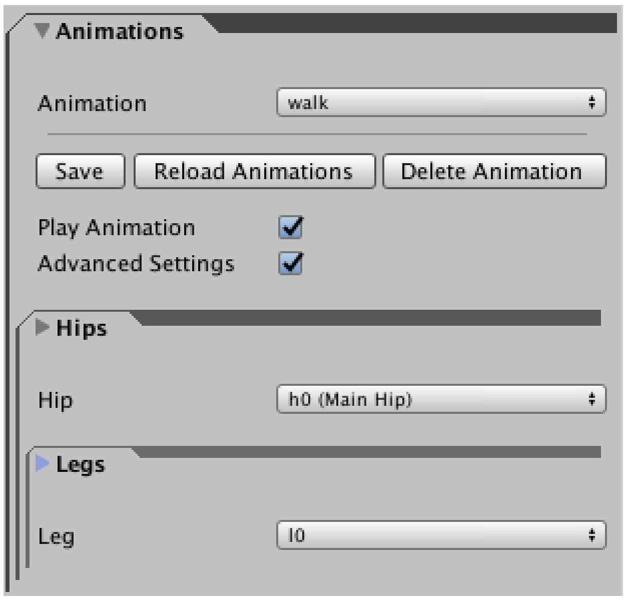
l0 Solver Target: None (Transform)
l1 Solver Target: None (Transform)

f0 (f1, f2...) Solver Target: Each IK solver target.

4.3.2 Animations

The Animation panel is enabled after the body is initialized. It holds the interface for Editing, Saving, and Loading the animations.

Note that Axial Bones and Legs are organized hierarchically under Animations, this is a reminder of the body hierarchy.


Animations

Animation: walk

Play Animation: ✓
Advanced Settings: ✓

Hips

Hip: h0 (Main Hip)

Legs

Leg: l0

Animation: The list for selecting one of the animations already saved in the body, or select “New Animation...” to edit and save a new animation.

OBS.: Once one animation is selected, it is automatically loaded on the body.

Save: Save the settings to the currently selected or new animation.

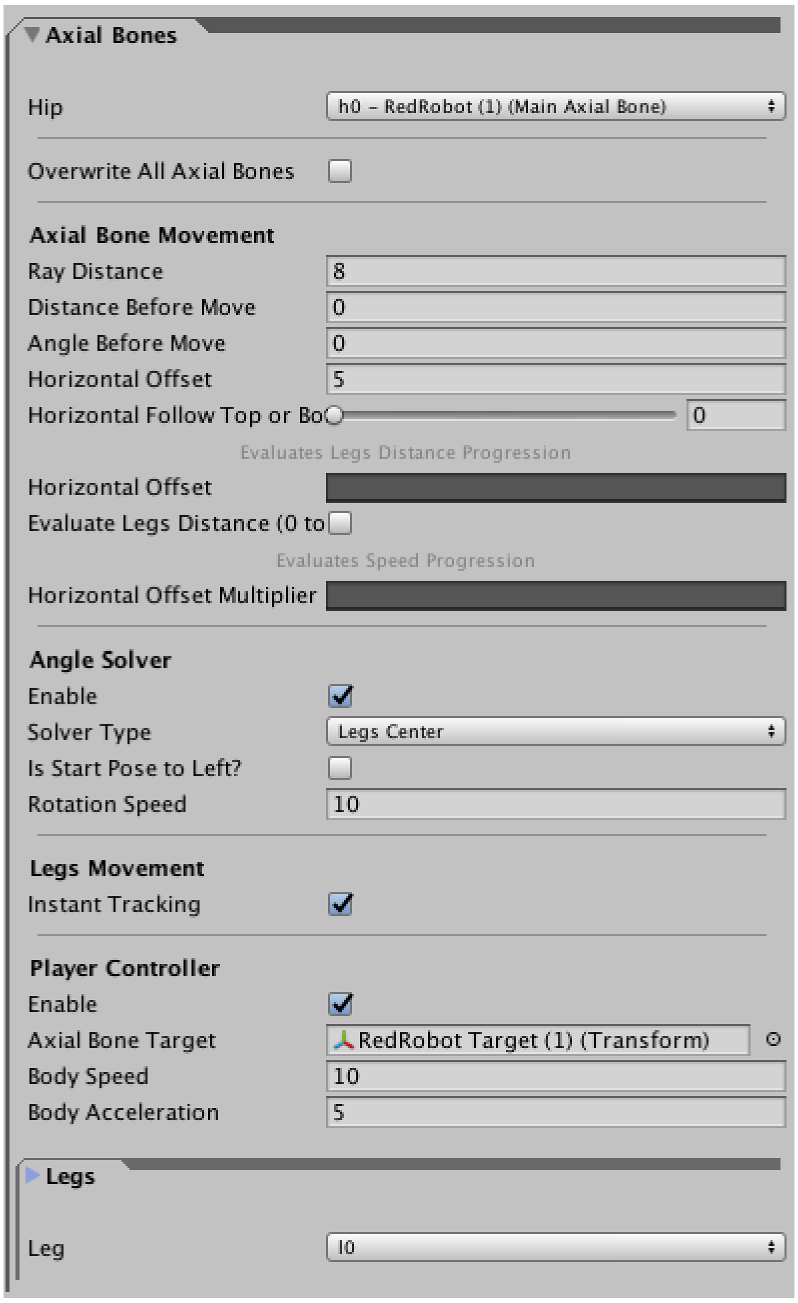
Delete Animation: Delete the currently selected animation.

Play Animation: Mark this checkbox to keep the animation running.

Advanced Settings: Show the advanced Axial Bones and Legs settings.

4.3.3 Axial Bones

This panel holds the settings regarding each Axial Bone of the body, these can be edited one by one independently, but it is also possible to copy the same values to all the axial bones by pressing the "Override All Axial Bones" button (See below further explanation on this button).



Axial Bones

Hip: h0 - RedRobot (1) (Main Axial Bone)

Overwrite All Axial Bones: ☐

Axial Bone Movement

Ray Distance	8
Distance Before Move	0
Angle Before Move	0
Horizontal Offset	5
Horizontal Follow Top or Bo	0

Evaluates Legs Distance Progression

Horizontal Offset

Evaluate Legs Distance (0 to ☐

Evaluates Speed Progression

Horizontal Offset Multiplier

Angle Solver

Enable	✓
Solver Type	Legs Center
Is Start Pose to Left?	☐
Rotation Speed	10

Legs Movement

Instant Tracking: ✓

Player Controller

Enable	✓
Axial Bone Target	RedRobot Target (1) (Transform)
Body Speed	10
Body Acceleration	5

Legs

Leg: l0

Axial Bone: List of the body's Axial Bones. Selecting one of them will display its current settings.

Override All Axial Bones: By pressing this button, the present values are copied to all the Axial Bones. Keep in mind that this option will override the possible edits you have made on other Axial Bones.

Axial Bone Movement

AutoSet Body Distance to Ground: Automatically set the distance from the Axial Bone to the ground.

Body Distance to Ground: Distance from the Axial Bone to the Ground.

AutoSet Ray to Ground Length: Automatically set the Length of the raycast, starting from the Axial Bone position directed down.

Ray to Ground Length: Length of the raycast, starting from the Axial Bone position directed down. This raycast checks if the body is grounded (the animation is activated) or not (the animation stops and legs are retracted to the rest position).

Distance Before Move Leg: Horizontal distance translated by the Axial Bone to activate one Leg step.

Angle Before Move Leg: Rotation in degrees of the Axial Bone, in relation to its center, to activate one Leg step.

Body Stabilization (Top Foot or Bottom Foot): This slider indicates If the horizontal position of the Axial Bone follows the most top or the bottom Foot. If value is 0, the bone follows the most top Foot, if value is 1, follows the most bottom. By selecting a value in between, the most top and bottom Feet positions are evaluated to indicate the position of the bone.

Evaluates Speed Progress

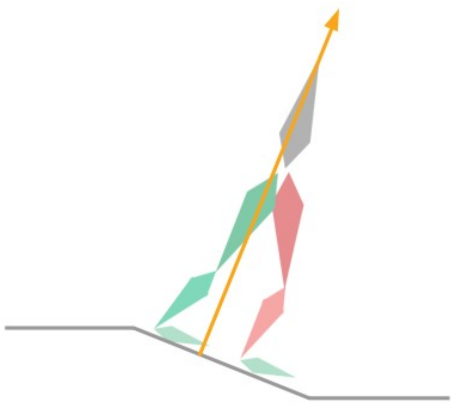
Vertical Offset Multiplier: Curve that evaluates the body speed. Its value multiplies the the current Vertical Offset value.

Angle Solver

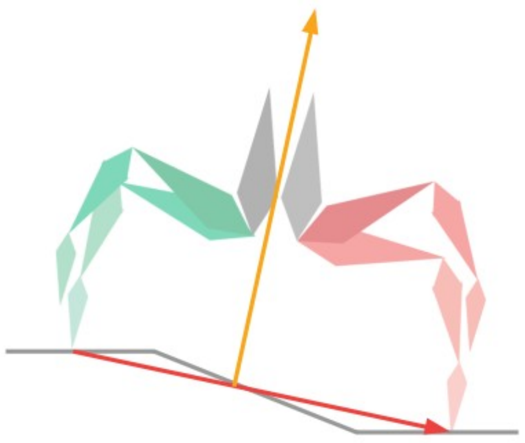
Enable: Enable rotation of the Axial Bone in relation to the ground surface.

Solver Type: List of angle solver types, “Raycast” or “Relative to Legs”.

Raycast: Rotates the body in relation to the ground normal.



Relative to Legs: Rotates the body in relation to the angle formed by the vector that goes from the bottom Foot of this Axial Bone to the bottom Foot of the next Axial Bone. If this is the last Axial Bone, then the angle is between this and the previous one.



Keep Vertical: Keeps the Axial Bone with Euler angle Z = 0.

Rotation Speed: Speed that the Axial Bone rotate towards the end angle evaluated by the angle solver.

Player Controller

Enable: Enables the PL2D Player Controller. It includes moving with “A” and “D” keys and jumping with “W”, it also have the possibility to add wall walk to the character.

Controller Type: List of controller types, “Keyboard Input” or “Path Follow”.

Keyboard Input: Enables body movement by using keys “A” (left) and “D” (right).

Path Follow: Enables body movement toward a point or a list of path points.

Body Speed: Maximum body speed.

Body Acceleration: Acceleration towards the maximum speed.

Wall Walk: Enables wall walk, making the player follow the surface angle.

Flip to Mouse: Enables body flip horizontally to face the mouse.

Invert Flip: Inverts horizontally the side the body is facing while using “Flip to Mouse”.

Jump: Enables jump using “W” key.

Jump Force: changes the jump force.

Pre Leg Bend Time: Duration of the pre jump, when the body lowers and the legs bend before the jump.

Add Point: Adds a path point to the bottom of the path points list.

Remove Point: Removes the last point of the path points list.

Point 0 (1, 2, 3…): path points represented by its Vector3 position.

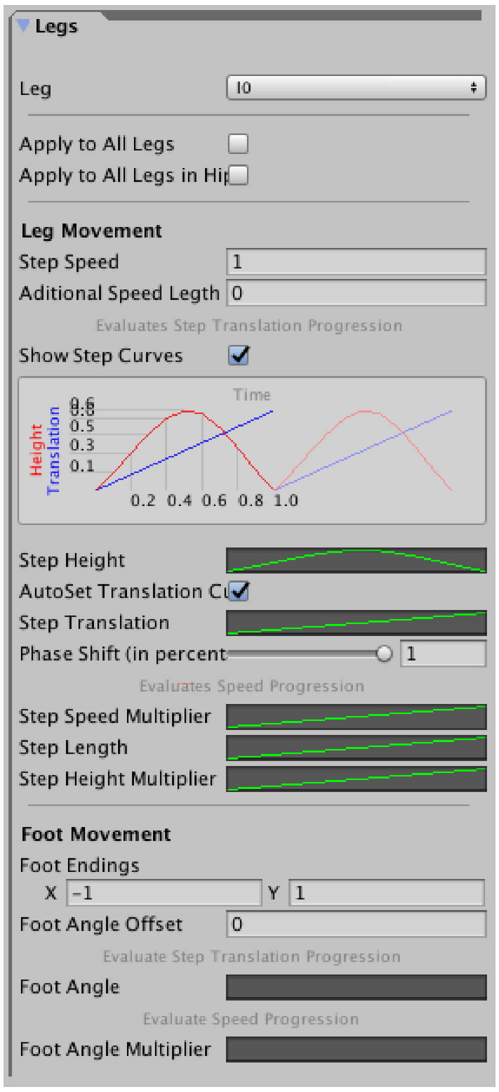
OBS.: The points position can be adjusted using their handles in the scene.

Play Path Follow: Starts the body movement following the path points.

Stop Path Follow: Stops the body movement of following the path points.

4.3.4 Legs

This panel holds the settings regarding each Leg in the selected Axial Bone, these can be edited one by one, however it is also possible to copy the same values to all the Legs on the selected Axial Bone or to all Legs on the Body by pressing the corresponding Override button. (See below further explanation of those buttons).



Legs

Leg	I0
Apply to All Legs	☐
Apply to All Legs in Hi	☐

Leg Movement

Step Speed	1
Aditional Speed Legth	0

Evaluates Step Translation Progression

Show Step Curves	✓
Step Height	
AutoSet Translation C	✓
Step Translation	
Phase Shift (in percent	1

Evaluates Speed Progression

Step Speed Multiplier	
Step Length	
Step Height Multiplier	

Foot Movement

Foot Endings X -1 Y 1

Foot Angle Offset	0

Evaluate Step Translation Progression

Foot Angle

Evaluate Speed Progression

Foot Angle Multiplier

The Legs movement is composed of horizontal translation (it will be called only Leg translation) and height translation.

The beginning of the step is the position of the Foot right before the Leg translation, the end position is the evaluated position of the Foot, taking in consideration the variables “Additional Step Length” and “Step Length” (curve) ad the Axial Bone’s position.

Leg: List of Legs in the selected Axial Bones. Selecting one of them will display its current settings.

Override All Legs in Axial Bone: By pressing this button, the present values are copied to all Legs in the selected Axial Bones. Keep in mind that this option will override the settings of all Legs in the selected Axial Bones by the viewed settings.

Override All Legs in Body: By pressing this button, the present values are copied to all Legs in the body. <u>Keep in mind that this option will override the settings of all Legs in the body by the viewed settings</u>.

Leg Movement

Step Speed: Speed of the Leg translation.

Additional Step Length: Increases (positive values) or decreases (negative values) the size of the step, adding the value to the forward horizontal length of the step.

Evaluate Step Translation Progress

Show Step Curves: Shows a <u>Height x Time</u> and <u>Translation x Time</u> graphs overlapped. It helps the user to visualize the step progression and adjust the settings.

Step Height: Curve that defines the height of the step over time, from the beginning to the end of the step.

AutoSet Translation Curve: Automatically creates the “Step Translation” (curve) by taking in consideration the first point > 0 and the next point <= 0 from the “Step Height” (curve), making the Translation start once the Foot leaves the ground to when it reaches the ground again.

Step Translation: Curve that defines the step translation over the step start and end points. Being the value 0 the start Foot position and the value 1 the end Foot position.

Phase Shift From Previous Step: The value indicates the previous step duration in percentage, being 0 the beginning of the last step and 1 the end of it.

By setting it 1, this Leg’s step will start right after the previous step. By decreasing from 1 to 0 it will make the steps overlap by making this Leg’s step start before the last step end.

Marking “Show Step Curves” will help see the end result.

Evaluate Speed Progress

Step Speed Multiplier: Multiplies the “Step Speed” by evaluating the body Speed.

Step Length: Adds a value to the Step Length by evaluating the body Speed.

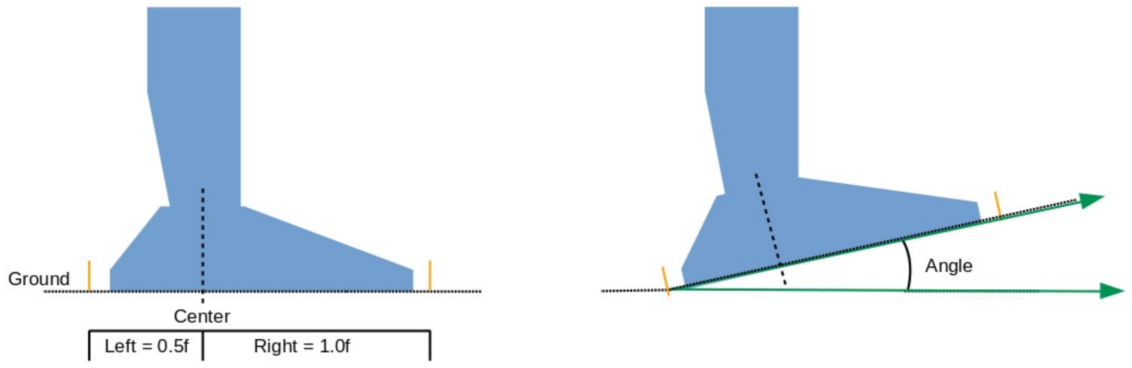
Step Height Multiplier: Multiplies the Step Height by evaluating the body Speed.

Foot Movement

Foot Endings: Used to indicate how the Foot will be angled in relation to the ground. See the image below for an example on how it is set up.

Left: Length of the left part (from the center) of the Foot.

Right: Length of the right part (from the center) of the Foot.



OBS.: The picture above is an example of “Foot Endings”, “Left” and “Right”, values that have a slight distance from the actual endings. This is how I personally set it to make the Foot Angle smoother on the ground and it is recommended for thin foots, despite that, feel free to set the values right in the endings if it fits better.

Foot Angle Offset: Angle offset in degrees. It offsets the default Foot angle.

Evaluates Step Translation Progress

Foot Angle: Adds the value in degrees by evaluating the Step Translation (from the start to end of the translation).

Evaluates Speed Progress

Foot Angle Multiplier: Multiplies the Foot Angle by evaluating the body Speed.

4.4 API

4.4.1 PL2D_Animator

The PL2D Animator can be accessed by script using its API. It is called from a reference of the desired PL2D Animator.

```
// animatorRef is the GameObject containing the  PL2D_Animator component
PL2D_Animator pl2dAnimator = animatorRef.GetComponent<PL2D_Animator>();
```

public string Initialize(Transform rootBone, SolverTypes solverType, bool clearIK = false, int chainLength = 0, Transform[] customSolverTargets = null)

public string LoadAnimation(string animationName)

public string SaveAnimation(string animationName)

public string DeleteAnimation(string animationName)

public void OverrideAllLegsInAxialBone(PL2D_Leg leg)

public void OverrideAllLegsInBody(PL2D_Leg leg)

public void ApplyDefaultSettings()

public List<PL2D_AxialBone> axialBones

4.4.2 PL2D_AxialBone

From the Axial Bone its possible to access its settings, Player Controller and child Legs

public bool AutosetBodyDistanceToGround

public float bodyDistanceToGround

public bool AutosetRayToGroundLength

public float rayToGroundLength

public float distanceBeforeMoveLeg

public float angleBeforeMoveLeg

public float bodyStabilizationTopBottomFoot

Vertical Offset Multiplier
public AnimationCurve multiplierlegsDistanceOnVPosCurveVI

public bool AngleSolverEnabled

public AngleSolverEnum angleSolverType

Rotation Speed
public float angleTrackSpeed

public bool PlayerControllerEnabled

public PlayerControllerEnum playerControllerType

public PL2D_PlayerController pl2d_PlayerController

```
// get the reference to the first axial bone in the list
PL2D_AxialBone axialBone0 = pl2dAnimator.axialBones[0];

// get pl2d player controller
PL2D_PlayerController pl2dPlayerController = axialBone0.pl2d_PlayerController
```

public List<PL2D_Leg> legs

```
// get the reference to the first axial bone in the list
PL2D_AxialBone axialBone0 = pl2dAnimator.axialBones[0];

// get legs and print
foreach(PL2D_Leg leg in axialBone0.legs)
{
    Debug.Log(leg);
}
```

4.4.3 PL2D_PlayerController

public float bodySpeed;

public float bodyAcceleration;

public bool enableWallWalk;

public bool enableFlipToMouse;

public bool invertFlipToMouse;

public bool enableJump;

public float jumpForce;

public float horizontalVelocityMultiplierForce;

public float preLegBendTime;

public void AddPathPoint(Vector3 position, int index)

public void RemovePathPoint(int index)

public void PlayPathFollow()

public void PausePathFollow()

4.4.4 PL2D_Leg

public float stepSpeed

public float additionalStepLength

public AnimationCurve stepHeightCurveSI

public bool AutoSetTranslationCurve

public AnimationCurve stepTranslationCurveSI

public float phaseShiftFromPreviousStep

Step Speed Multiplier
public AnimationCurve multiplierSpeedCurveVI;

Step Length
public AnimationCurve additionalStepLengthCurveVI;

Step Height Multiplier
public AnimationCurve multiplierHeightCurveVI;

public Vector2 footEndings

public float footAngleOffset

public AnimationCurve footAngleCurveSI

Foot Angle Multiplier
public AnimationCurve multiplierAngleCurveVI;

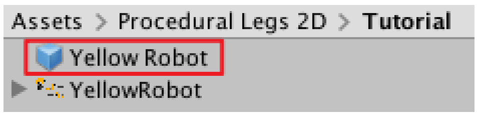
5 STEP BY STEP SETUP GUIDE (QUADRUPED)

In this section there is a quick guide on setting up the body and an walk animation for a quadruped body using the PL2D Animator interface.

5.1 SET UP THE BODY

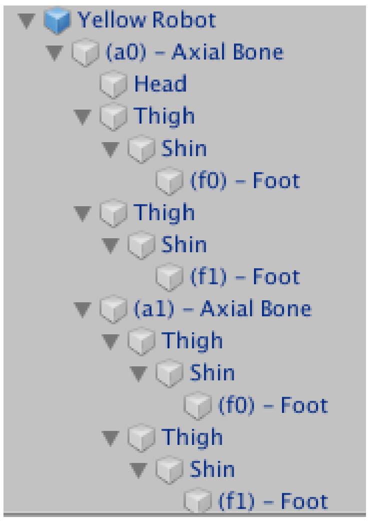
The body rigging is done by hierarchically setting up the GameObjects that form the body as shown in the section "Body Rigging"

For this tutorial, there is a rigged body already setup named "Yellow Robot" as prefab in the "Example Scenes > Tutorial > Prefabs" folder.



Assets > Procedural Legs 2D > Tutorial
Yellow Robot
YellowRobot

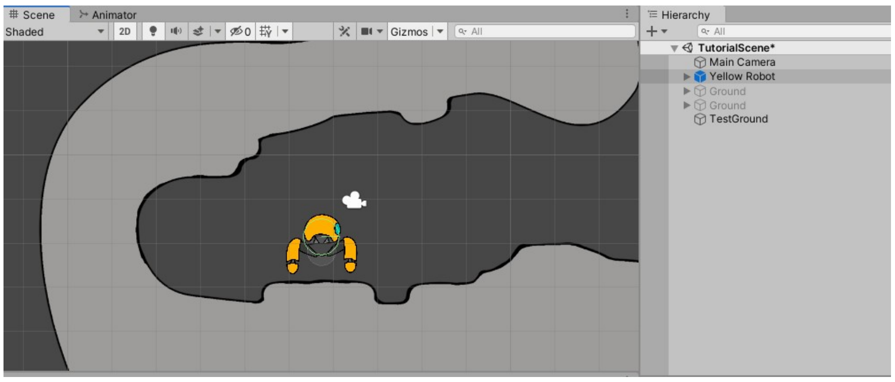
1.



```
Yellow Robot
  (a0) – Axial Bone
    Head
    Thigh
      Shin
        (f0) – Foot
    Thigh
      Shin
        (f1) – Foot
    (a1) – Axial Bone
      Thigh
        Shin
          (f0) – Foot
      Thigh
        Shin
          (f1) – Foot
```

1. **Drag the Yellow Robot prefab to the Scene**. You can see its hierarchy structure as an example for other bodies you want to create.

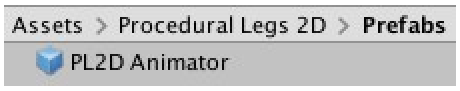
Reminder: The Axial Bones and Foots must be correctly named so the PL2D Animator can identify them. Axial Bones must begin with "a" followed by the index (0, 1, 2...) wrapped in round brackets. Ex.: "(a0)", "(a1)" etc. Legs must begin with "f" followed by the index (0, 1, 2...) wrapped in round brackets. Ex.: "(f0)", "(f1)" etc. The legs must be child of an Axial Bone.

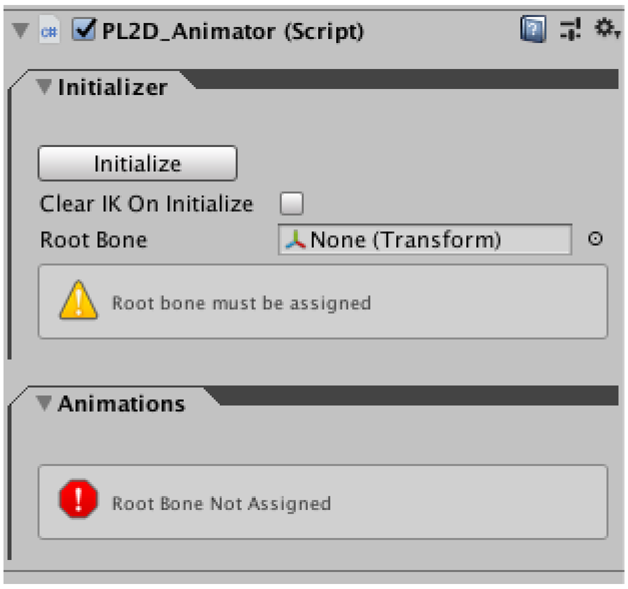


5.2 Initializing the PL2D Animator

Once you have your rigged body on the scene, it needs a PL2D Animator in order to be connected with the system and be able to receive and run the animations.

The PL2D Animator GameObject can be found on the "Prefabs" folder.


Assets > Procedural Legs 2D > Prefabs
PL2D Animator

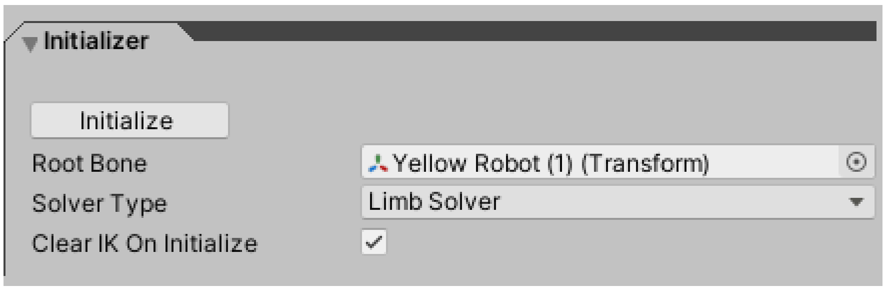

PL2D_Animator (Script)
Initializer
Initialize
Clear IK On Initialize
Root Bone None (Transform)
Root bone must be assigned
Animations
Root Bone Not Assigned

1. **Drag the PL2D Animator prefab to the scene** and **change its name to "Robot - PL2D Animator"**. It will show the interface for Initializing the body and later configure the animations.

<u>Reminder:</u> To better organize your project, each animator must have an unique name.

2. We must select the body that will be initialized, for that, **set the Yellow Robot GameObject on the scene to the Root Bone field**, after that, the Initialization options will show up.

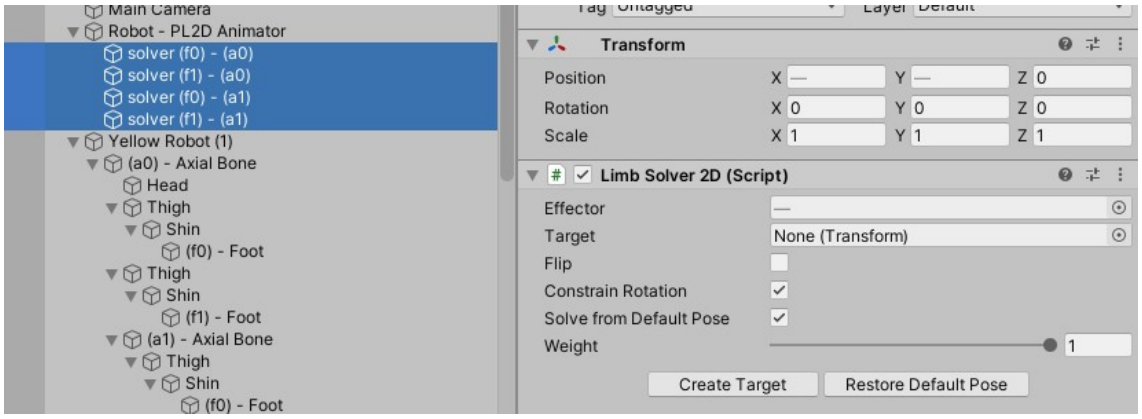
3. To finish the body initialization, **choose Limb Solver and click on "Initialize"**.


Initializer
Initialize
Root Bone Yellow Robot (1) (Transform)
Solver Type Limb Solver
Clear IK On Initialize ✓

By now, the system should have created a folder to store this particular animator's animations, "Assets > Procedural Legs 2D > Resources > PL2D Animations > Robot - PL2D_Animator".

Since we are using the Unity's IK system, we need to set up the IK solvers, that are created as children of the animators, by positioning them to the point where the feet touch the ground and create the targets.

4. **Select and position all the solvers and click on "Create Target"**. Note that some legs will flip in a wrong angle, to correct that, **select those solvers and check the "Flip" box**.



Main Camera
Robot - PL2D Animator
solver (f0) - (a0)
solver (f1) - (a0)
solver (f0) - (a1)
solver (f1) - (a1)
Yellow Robot (1)
(a0) - Axial Bone
Head
Thigh
Shin
(f0) - Foot
Thigh
Shin
(f1) - Foot
(a1) - Axial Bone
Thigh
Shin
(f0) - Foot

Tag Untagged Layer Default

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

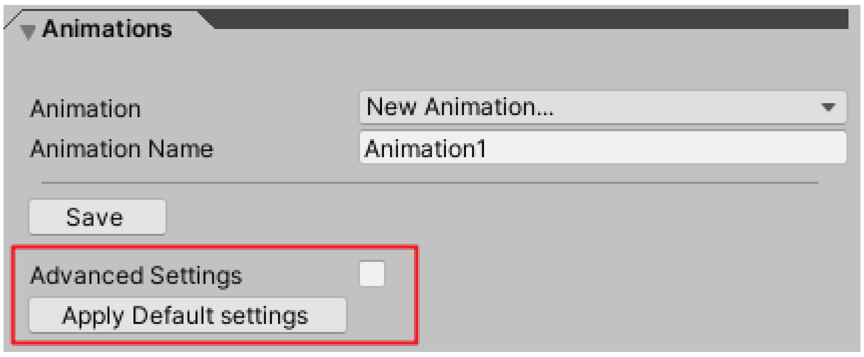
Limb Solver 2D (Script)
Effector —
Target None (Transform)
Flip ☐
Constrain Rotation ✓
Solve from Default Pose ✓
Weight 1

5.3 Setting Up Axial Bones

For this setup to be quick and easy, lets hide some of the variables we are not going to use right now and use the default settings.

1. In the top of the Animations panel, make sure the **"Advanced Settings" is turned off**.

2. Now let's **click on "Apply Default Settings"** so it give us some default values to start with.



Animations
Animation New Animation...
Animation Name Animation1
Advanced Settings ☐

By now, the Axial Bones and Legs tabs are showing up and a new animations is ready to be created.

The main Axial Bone, “(a0)”, is already selected, this is the bone that two of our robot’s legs are attached and also have another axial bone as child, it contains some options that are used by the whole body, these values are disabled if you choose another Axial Bone.

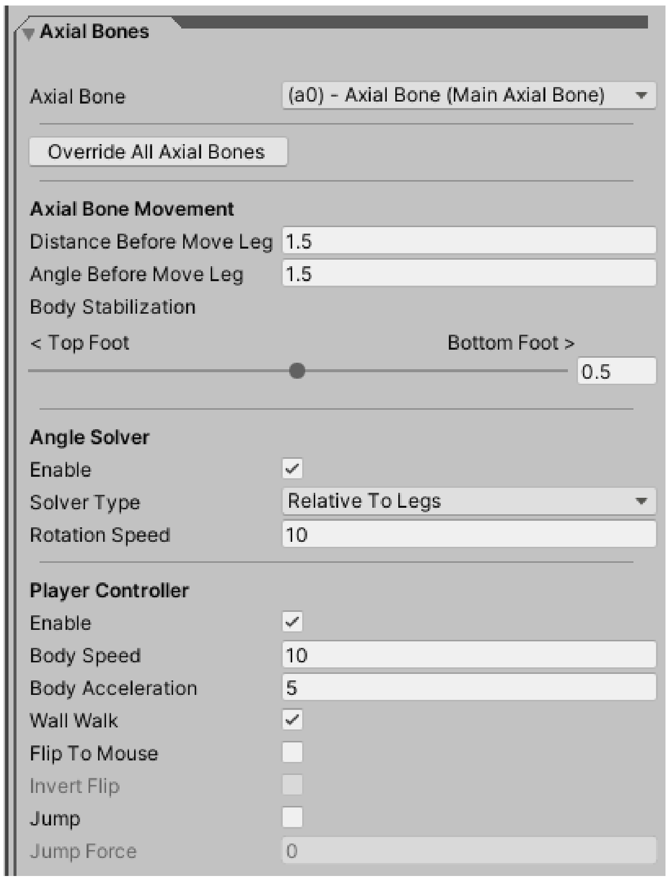
3. Then we are going to **Enable the Angle Solver and choose the type “Relative to Legs”** and **set the “Rotation Speed” = 10**. This will enable the body to follow the ground surface angle.

4. We are also going to **Enable the Player Controller**, and make sure the **type is “Keyboard Input”, the “Body Speed” = 10** and **“Body Acceleration” = 5**. This will enable the player to move the body by using the keys “A”, “D”.

5. **Check the “Wall Walk” box** to enable the robot to craw over walls .

6. To finish, we are going to repeat this options to all the Axial Bones, for that, **press the “Override All Axial Bones” button**.

End Result:



Axial Bones

Axial Bone	(a0) - Axial Bone (Main Axial Bone)

Override All Axial Bones

Axial Bone Movement

Distance Before Move Leg	1.5
Angle Before Move Leg	1.5
Body Stabilization	
< Top Foot / Bottom Foot >	0.5

Angle Solver

Enable	✓
Solver Type	Relative To Legs
Rotation Speed	10

Player Controller

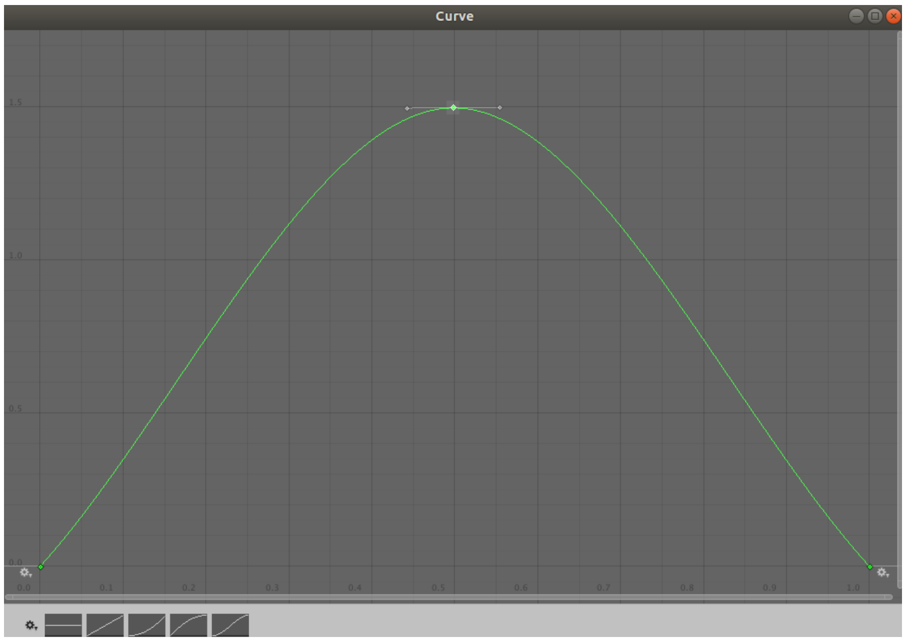
Enable	✓
Body Speed	10
Body Acceleration	5
Wall Walk	✓
Flip To Mouse	
Invert Flip	
Jump	
Jump Force	0

5.4 Setting Up Legs

Any Leg can be selected, since we want all the Legs to have the same values.

1. We adjust the step speed, **set the "Step Speed" = 4**.

<u>Reminder</u>: You can play with the values of the "Step Height" curve and see how it changes the step behavior. Keep it default to follow this quick tutorial.



This curve has three points:

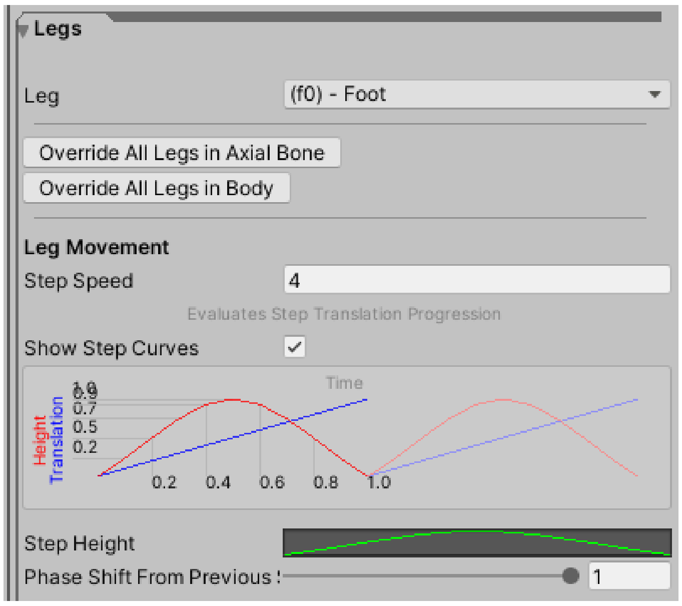
- *(0, 0)*
- *(0.5, 1.0)*
- *(1, 0)*

OBS.: Optionally, you can mark the option "Show Step Curves" just to visually evaluate the curves.

Note that "Phase Shift From Previous Step" is equal 1, this delays the next step from the previous one by shifting the step curves.

2. To finish setting up the legs, let's copy the same values to the other Legs, **press the "Override All Legs in Body" button**.

End result:


Legs

Leg: (f0) - Foot

Override All Legs in Axial Bone

Override All Legs in Body

Leg Movement

Step Speed: 4

Evaluates Step Translation Progression

Show Step Curves: ✓

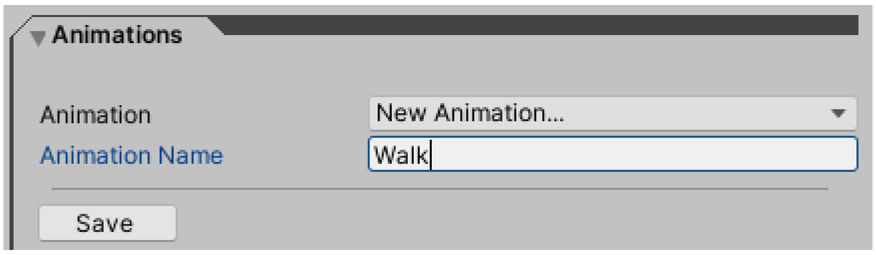
Step Height

Phase Shift From Previous: 1

5.5 Saving and Testing

We can now test our animation.

1. To save this animation, scroll up to the beginning of the Animations panel and **change the "Animation Name" to "Walk"** and **save it clicking on the "Save" button**.


Animations

Animation: New Animation...

Animation Name: Walk

2. The animation will be stored in the proper folder and we can **Play the scene and use the buttons "A" and "D" to move the Robot**.

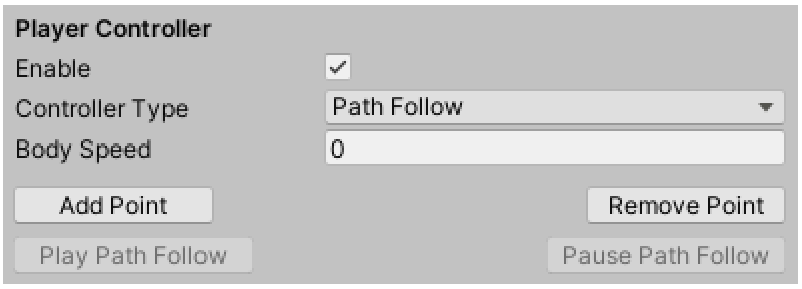
6 ANNEX

Explanation and usage of useful features of the asset.

6.1 Using the "Path Follow" Controller Type

The "Path Follow" type let you create a sequence of points for the body to follow. The Legs movement will behave accordingly following the body speed and relative to ground position.

Having the "Player Controller" enabled, once you select "Path Follow" as the Controller Type on the Animator Inspector, the interface will present buttons for adding and removing points of the path points list.



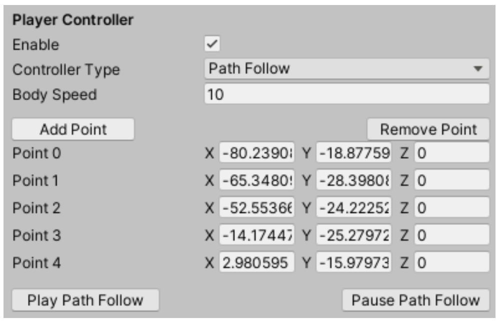
Player Controller

Enable	✓
Controller Type	Path Follow
Body Speed	0

The buttons for playing and pausing the path follow will be enabled with at least one point added.

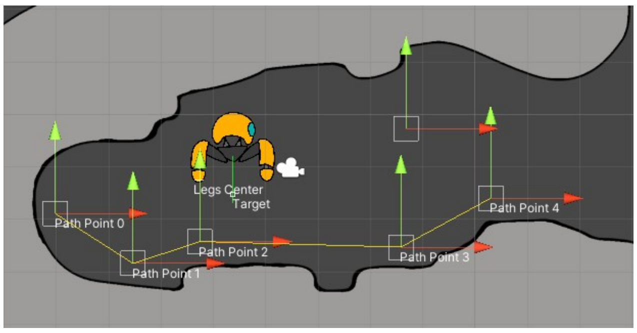
The points added to the list will be shown by a handle and label in the scene (on Editor or Play modes) with yellow lines between each "n" and "n+1" points.

You can drag the handles to adjust the path.



Player Controller

Enable	✓					
Controller Type	Path Follow					
Body Speed	10					
Point 0	X	-80.2390	Y	-18.87759	Z	0
Point 1	X	-65.3480	Y	-28.3980	Z	0
Point 2	X	-52.5536	Y	-24.2225	Z	0
Point 3	X	-14.1744	Y	-25.2797	Z	0
Point 4	X	2.980595	Y	-15.97973	Z	0



Press "Play Path Follow" to make the body start moving towards the first point in the sequence, when it reaches the point, follow the next one. When the body reaches the last point it starts targeting the previous points until the first point again.

Pausing the path follow will make the body stop the movement and keep the current direction and target point.